COLLECTION Ingénieurs E.E.A. (1) (Électronique, Électrotechnique, Automatique).
A l'usage de l'Enseignement Supérieur : Écoles d'Ingénieurs, U.E.R., CNAM...

ÉLECTRONIQUE

Cours d'Électronique, (en cinq volumes), par Françis MILSANT :

Tome I. — *Circuits à régime variable* (1981).
Tome II. — *Composants électroniques* (1981).
Tome III. — *Amplification. Circuits intégrés* (1982).
Tome IV. — *Contre-réaction, oscillation, transformation de signaux* (1980).
Tome V. — *Diodes, thyristors, commande des moteurs* (1981).

Problèmes d'Électronique, par Françis MILSANT.

Tome I. — *Circuits à régime variable* (1980).
Tome II. — *Composants électroniques* (1982).
Tome III. — *Amplification. Circuits intégrés* (1982).

Mesures d'Électronique, par Henri CATELIN et Pierre JOUBERT :

Tome I. — *Circuits à régime variable* (1973).
Tome II. — *Composants électroniques* (1977).

Électronique de puissance

1. *Commande des moteurs à courant continu*, par Robert CHAUPRADE (1981).
2. *Commande des moteurs à courant alternatif*, par Robert CHAUPRADE et Françis MILSANT (1980).

ÉLECTROTECHNIQUE

Cours d'Électrotechnique (en 3 volumes), par Françis MILSANT :

Tome I. — *Machines à courant alternatif* (en préparation).
Tome II. — *Machines à courant continu* (1981).

Problèmes d'Électrotechnique avec leurs solutions, par Michel BORNAND :

— *Moteurs à courant continu et leur commande par thyristor* (1980).
— *Machines en courant alternatif et électronique de puissance* (1980).

Générateurs, lignes, récepteurs, par C.E. MOORHOUSE, traduit de l'anglais par Françis MILSANT (1977).

AUTOMATIQUE

Cours de physique des vibrations, par A. FOUILLÉ et P. DÉRÉTHÉ (1977).

Asservissements linéaires, par Françis MILSANT :

Tome I. — *Analyse* (1981).
Tome II. — *Synthèse* (1979).

Automatismes à séquences, par Maurice MILSANT (1979).

Pratiques Séquentielles et Réseaux de Pétri, par Sylvain THELLIEZ (1978).

Microprocesseurs à l'usage des électroniciens. Structure et fonctionnement, par J.P. COCQUEREZ et J. DEVARS (1980).

COLLECTION INGÉNIEURS E.E.A.
sous la direction de Francis MILSANT

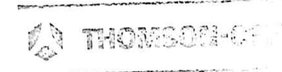
GRAFCET
ET LOGIQUE INDUSTRIELLE PROGRAMMÉE

par

Sylvain THELLIEZ
Professeur
à l'ENSAM de Cluny

Jean-Marc TOULOTTE
Professeur
à l'Université de Lille

Deuxième édition
NOUVEAU TIRAGE





61, Boulevard Saint-Germain - 75005 Paris
1982

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Editions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

AVANT-PROPOS

Ce livre constitue un exposé illustré à l'aide d'exercices des méthodes de synthèse et de matérialisation des structures de commande en logique industrielle programmée.

Il comporte quatre parties principales ; ce sont :

— l'examen de quelques méthodes d'implantation des fonctions combinatoires sur matériel programmé.

— la définition, l'étude et l'utilisation de deux outils graphiques de description : le GRAFCET et les réseaux de pétri.

— les méthodes d'implantation des outils de description compte tenu des caractéristiques fondamentales du matériel choisi par le concepteur.

— l'utilisation, basée sur une application directe des méthodes d'implantation des outils de description, des dispositifs spécifiques de la logique cablée programmée, des automates programmables, des machines universelles ou des microprocesseurs.

Il s'adresse aux étudiants de l'enseignement supérieur et aux techniciens et ingénieurs de l'industrie qui souhaitent se familiariser avec l'emploi des méthodes de la logique programmée.

TABLE DES MATIÈRES

I. LA FONCTION COMBINATOIRE

Tous les ouvrages de logique classique traitent de la fonction combinatoire. Celle-ci est effectivement importante car elle intervient à tous les niveaux dans la mise en œuvre d'un automatisme logique. Suivant les habitudes des utilisateurs, elle peut être plus ou moins complexe et revêtir des formes très variées. Ce chapitre a pour but d'examiner quelques méthodes d'implantation des fonctions combinatoires sur matériel programmé.

I.1. Les diverses formes de fonction combinatoire

Suivant les traitements effectués, une fonction combinatoire se rencontre classiquement sous trois formes algébriques : la première forme canonique, l'écriture simplifiée en somme d'intersections premières (ou implicants premiers) et une forme avec mise en facteur. On aurait ainsi pour la table de vérité de la figure I.1. :

$$F = a\bar{b}c + ab\bar{c} + abc + \bar{a}bc \qquad \text{(I.1)}$$

$$F = ab + ac + bc \qquad \text{(I.2)}$$

$$F = a(b + c) + bc \qquad \text{(I.3)}$$

a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Fig. I.1

D'autres écritures issues des diverses formes canoniques sont évidemment possibles. Dans les cas les plus complexes, la fonction se présente comme une séquence de variables logiques, d'opérateurs tels que OU, ET, OU exclusif et de parenthèses.

Par ailleurs, dans de nombreux problèmes, il convient de considérer le traitement simultané d'un ensemble de fonctions logiques des mêmes variables.

Une autre présentation possible des fonctions logiques est celle des tables de décision. Celles-ci sont en fait des tables de vérité où figurent les combinaisons de condition qui donnent un résultat au niveau des actions. De plus, les actions peuvent éventuellement être indicées avec un ordre de mise en application. Toutes les méthodes pour fonctions multiples, où l'application ne se fait pas globalement, sont donc utilisables.

I.2. Les méthodes de matérialisation

1.2.1. L'arbre de décision logique

Il s'agit d'une représentation comprenant deux types d'éléments :

— des assignations de valeur aux fonctions,

— des tests portant sur une des variables et indiquant, suivant sa valeur, le chemin à suivre pour aller du point de départ de l'arbre jusqu'à un point d'assignation.

L'élément de test est donc un aiguillage piloté par la variable testée. Par convention, si la valeur de la variable est égale à zéro, la sortie de l'élément de test est indiquée par un petit cercle de complémentation, si elle est égale à 1, elle correspond à l'autre sortie.

Un arbre booléen comporte un et un seul point de départ et il ne peut y avoir qu'une seule liaison entre une sortie d'un test et l'entrée de l'élément suivant.

La fonction (I.1) précédente se représente sous la forme de l'arbre de la figure I.2).

Certaines branches mènent à un résultat identique, ce qui permet de condenser cet arbre sous la forme de la figure I.3.

Il est facile de constater que sur un arbre de décision binaire, tous les monômes sont disjoints car à un moment on a dû passer par des voies de sortie différentes d'un même test. Les équations de la forme (I.2) posent alors des difficultés de représentation. Dans ce cas, il convient de se ramener à une forme utilisable.

La procédure pour rendre disjoints deux monômes booléens m_1 et m_2 d'une expression consiste à écrire

$$F = m_1 + m_2 + m_3 + \dots = m_1 + m_2.\overline{m_1} + m_3 \dots.$$

Il est parfois nécessaire de répéter la procédure au niveau des monômes créés. Ainsi

$$f = a.b + c.d. = ab + cd\,(\overline{ab}) = ab + \bar{a}cd + \bar{b}cd$$

donne des monômes $\bar{a}cd$ et $\bar{b}cd$ non disjoints.

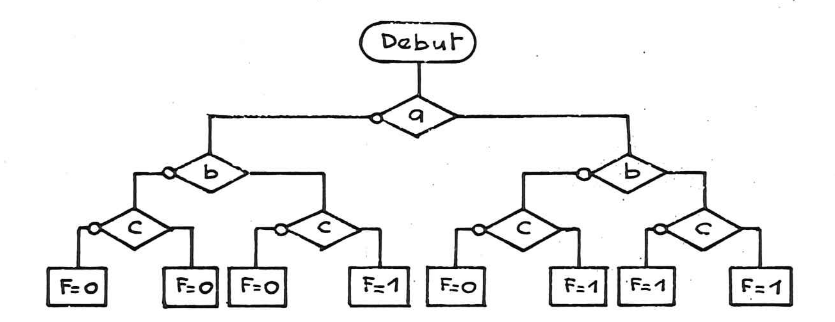


Fig. I.2

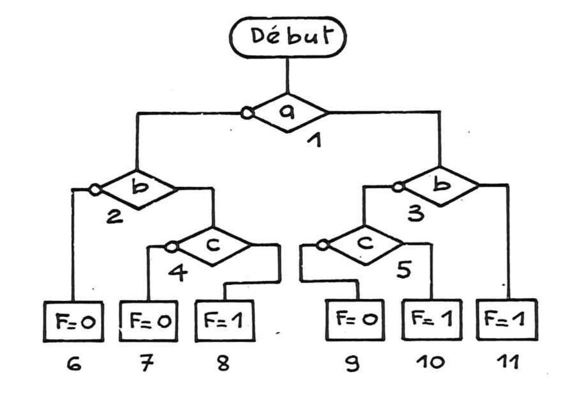


Fig. I.3

Il convient donc de reprendre :

$$f = ab + \bar{a}cd + \bar{b}cd(\overline{\bar{a}cd}) = ab + \bar{a}cd + a\bar{b}cd \qquad (I.4)$$

Suivant ce principe, la forme (I.2) devient :

$$F = ab + ac\,(\overline{ab}) + bc\,(\overline{ab + ac})$$

ce qui donne en développant :

$$F = ab + a\bar{b}c + \bar{a}bc \qquad (I.5)$$

Cette écriture permet alors d'obtenir directement la représentation de la figure I-3. Une autre façon de procéder à partir de la formule (I-3) est de considérer après chaque test la fonction résultante. Ainsi en commençant par le test de *a :* si $a = 0$, F devient *égal* à bc, si $a = 1$, $F = b + c$.

On en déduit les tests suivants et on construit le graphe de proche en proche.

L'implantation sur machine nécessite le passage au graphe de décision binaire. Pour cela, sur l'arbre de la figure I-3, on effectue une numérotation des éléments, puis on regroupe ces éléments par niveau. Le niveau *0* est celui des assignations.

$$\text{niveau } 0 = \{ 6, 7, 8, 9, 10, 11 \}$$

Le niveau 1 réunit le ou les tests portant sur une même variable et dont les deux alternatives sont reliées à des éléments du niveau 0.

$$\text{niveau } 1 = \{ 4,5 \}$$

Le niveau i est composé du ou des tests portant sur une même variable dont les deux alternatives sont reliées à des niveaux inférieurs à i.

$$\text{niveau } 2 = \{ 2,3 \}$$

$$\text{niveau } 3 = \{ 1 \}$$

Des regroupements peuvent se faire en appliquant la procédure suivante :

— Regroupement des éléments du niveau 0 d'assignation identique.

— Regroupement des tests de même niveau ayant des successeurs compatibles, c'est-à-dire appartenant à un même regroupement.

L'application de la procédure se fait à partir du niveau 0 vers les niveaux croissants. Sur notre exemple on obtient ainsi en utilisant une nouvelle numérotation des éléments

$$\{ 6^* \} = \{ 6,7,9 \} ; \{ 8^* \} = \{ 8,10,11 \} ; \{ 4^* \} = \{ 4,5 \}$$

ce qui donne le graphe de la figure I-4.

Dans la pratique, le traitement débute par la prise en compte de la valeur des variables logiques. Cette mesure, dans certains cas, se fait à l'appel de la variable lors du test. Puis une fois la valeur de *F* obtenue, on passe à la suite du programme ou on reboucle sur le début.

Le traitement des fonctions simultanées ne pose aucun problème si celles-ci sont disjointes, car on est ramené au cas précédent. Si elles ne le sont pas, il faut alors considérer les produits de fonction. Ceci sera illustré par l'exemple suivant :

$$F_1 = a\bar{d} + b + \bar{a}\,\bar{c}$$

$$F_2 = \bar{a}\,\bar{b} + cd$$

On établit les quatre produits :

$$P_1 = F_1.F_2 = bcd + \bar{a}\,\bar{b}\,\bar{c}$$

$$P_2 = F_1.\bar{F_2} = a\bar{d} + b\bar{c} + b\bar{d} = b\bar{c} + bc\bar{d} + a\bar{b}$$

$$P_3 = \bar{F_1}.F_2 = \bar{a}\bar{b}c + \bar{b}cd = \bar{b}cd + \bar{a}\bar{b}c\bar{d}$$

$$P_4 = \bar{F_1}.\bar{F_2} = a\,\bar{b}\,\bar{c}\,d$$

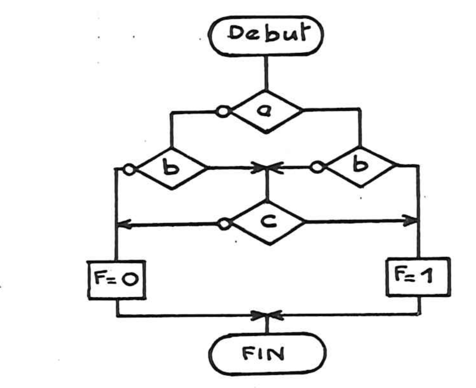


Fig. I.4

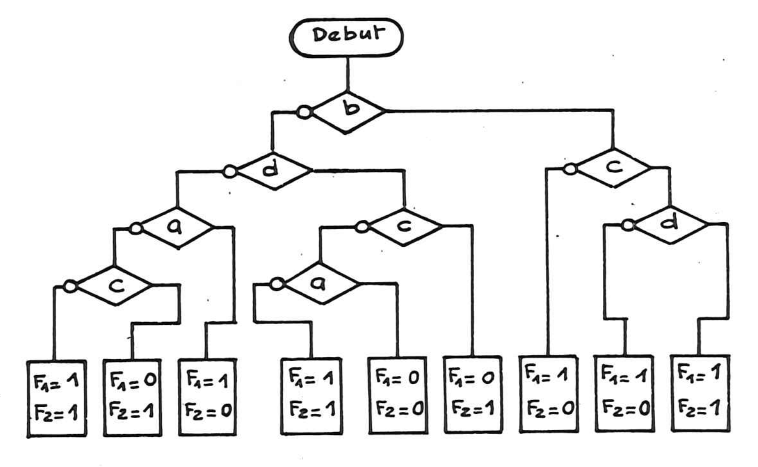


Fig. I.5 a)

Ils se représentent par exemple par l'arbre de la figure I.5 a). Le nombre de tests dépend du choix de l'ordre des tests. Il est intéressant pour diminuer le nombre de tests, de commencer par la variable présente sous forme normale et complémentée dans le maximum de produit *Pi*, puis de répéter cette procédure au niveau des fonctions résultant du choix de la première

variable en opérant de la même manière. Des méthodes heuristiques basées sur cette procédure ont été proposées, leur application aux fonctions *F1* et *F2* précédentes, conduit par exemple à l'arbre de la figure I.5 b) qui est à comparer à celui de la figure I.5 a.

De la même manière, dans le cas de 3 fonctions F_1, F_2, F_3 non disjointes, l'arbre de décision binaire s'obtient en considérant les $2^3 = 8$ produits $F_1F_2F_3$, $\overline{F_1}F_2F_3$,......, $\overline{F_1}\,\overline{F_2}\,\overline{F_3}$.

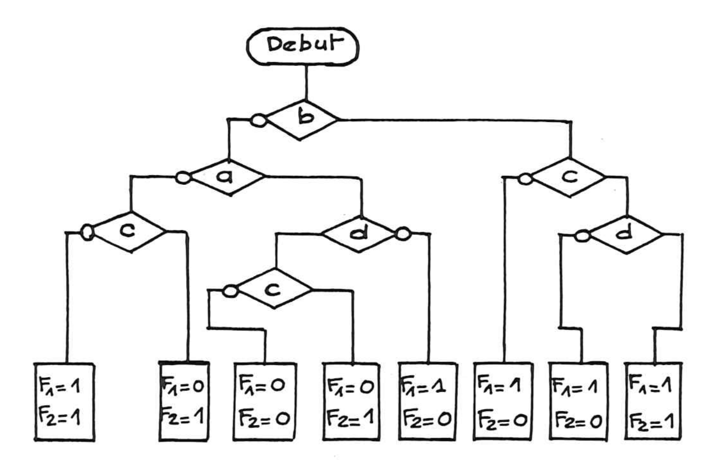


Fig. I.5 b)

Cette méthode est particulièrement bien adaptée aux machines disposant d'instructions de branchement conditionnel, et surtout à celles ayant l'instruction SI-ALORS-SINON (IF-THEN-ELSE). Elle est réellement puissante si on peut faire les traitements sur bit, c'est-à-dire si on a accès direct à une variable logique d'entrée. Elle présente un encombrement mémoire réduit et une vitesse excellente. Ses inconvénients majeurs sont liés à la structure «orientée programme» de l'implantation en machine ; car la taille du programme dépend de la fonction, une modification nécessite une refonte totale du programme, enfin la maintenance est plus difficile. Cette implantation s'oppose à celle «orientée données» où la fonction à représenter est mise dans une table de données gérée par un programme fixe.

1.2.2. L'organigramme

L'organigramme s'adapte à n'importe quel type de fonction écrite sous forme condensée ou non, avec des monomes disjoints ou non.

Soit par exemple la forme (I.3) : $F = a(b+c) + bc$.

Compte tenu de la simplicité de la fonction, on retrouve pour organigramme le graphe de décision binaire de la figure I-4.

Pour l'obtenir, le principe est de considérer que dans une fonction OU si une variable est à 1, la fonction vaut 1. Pour une fonction ET, la fonction vaut 0 si une variable vaut 0.

Pour des fonctions complexes, le résultat obtenu n'est en général pas optimal au sens de la minimisation du nombre de tests, car on peut être amené à faire plusieurs fois le même test dans une même branche.

Pour rendre le traitement plus efficace, on peut envisager une hiérarchisation dans l'équation en fonction du niveau d'apparition des variables. Par exemple pour la fonction $F = (\bar{A}+\bar{B})\,(C+D)\,E$, l'arbre de la figure I.6 indique l'ordre le plus intéressant pour faire le traitement ; le test à *0* de E est pris en premier, puis dans un ordre quelconque les autres tests peuvent être effectués car les variables se retrouvent au même niveau. On peut voir que si *E* vaut *0*, le traitement des autres tests est supprimé et la procédure plus rapide.

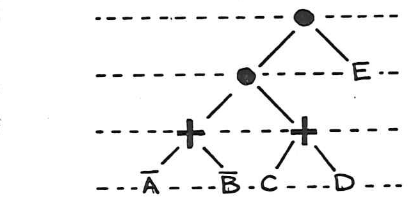


Fig. I.6

Le traitement des fonctions simultanées est plus problématique car il s'agit de trouver le maximum de tests communs pour avoir un résultat intéressant.

Cette méthode est plus directe à mettre en œuvre que la précédente. Elle est moins optimale, surtout pour les fonctions multiples. Elle présente les mêmes avantages et inconvénients.

Il est à noter que sur certaines machines, une variante de la méthode ne fait que des tests sur monôme, car les produits se font automatiquement.

1.2.3. Méthode par masquage

Cette méthode est particulièrement intéressante lorsque les variables sont à l'acquisition groupées par mot. Elle consiste à tester individuellement chaque monôme après passage du mot représentatif des variables d'entrée à travers un masque pour la sélection des variables du monôme. Soit la fonction (I.2) :

$$F = ab + ac + bc$$

et supposons les variables a, b, c placées dans cet ordre au début d'un mot regroupant huit variables (huit bits).

Cette fonction se transcrira sous la forme d'une table :

DEBTAB	1	1	0	0	0	0	0	0	(Masque 1)
	1	1	0	0	0	0	0	0	(Test 1)
	1	0	1	0	0	0	0	0	(Masque 2)
	1	0	1	0	0	0	0	0	(Test 2)
	0	1	1	0	0	0	0	0	(Masque 3)
FINTAB	0	1	1	0	0	0	0	0	(Test 3)

Cette structure de données est gérée par un programme de traitement fixe qui fait l'évaluation de la fonction. Après avoir figé dans MOTENT la valeur des variables d'entrée, il fait, pour le premier monôme, la sélection des variables a et b en effectuant un ET bit à bit avec le masque 1, puis par un OU exclusif bit à bit avec le mot test 1, on regarde si *a* et *b* valent 1. Dans ce cas le mot R résultant du OU exclusif est identique au mot nul 0 0 0 0 0 0 0 0 dénoté 0 dans l'ordinogramme de la figure I.7 et la fonction vaut 1 sinon on examine de la même façon le monôme suivant et ceci jusqu'à FINTAB. Le test est ici équivalent au masque car il n'y a pas de variables complémentées dans le monôme.

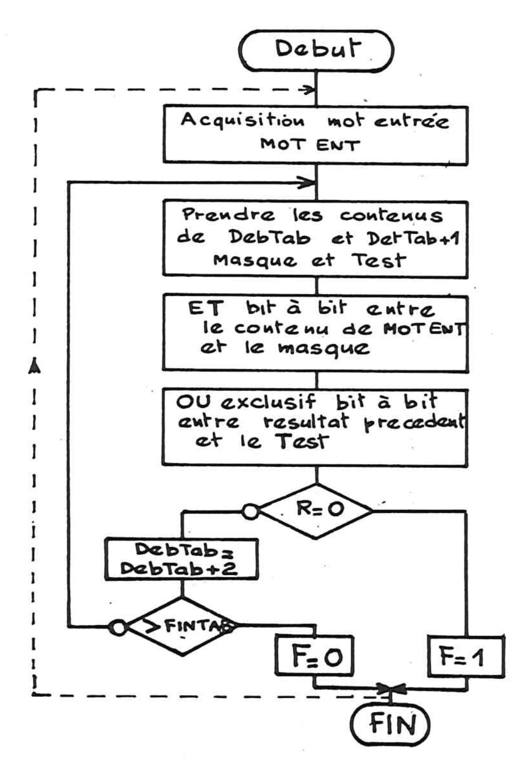


Fig. I.7

L'allure générale du programme est donnée par la figure I.7. Celui-ci peut s'insérer dans une structure plus vaste, permettant d'évaluer successivement plusieurs fonctions, respectivement définies par les adresses de début et de fin de table.

L'utilisation d'un programme fixe associé à une structure de données dépendant de la fonction est particulièrement intéressante pour effectuer la maintenance. En effet, il suffit pour cette opération de remplacer la fonction par une table particulière de test.

Quand le nombre de variables est important, il faut travailler sur plusieurs mots d'entrée. Ceci alourdit la procédure de traitement de chaque monôme ainsi que sa représentation dans la table des données. Un monôme examiné implique une valeur 1 de la fonction si les tests sur l'ensemble des mots sont satisfaits. Le nombre de mots peut être fixé et égal au nombre de mots d'entrée, certains mots pouvant alors être totalement indifférents. On peut aussi ne retenir que les mots significatifs en débutant le monôme dans la table des données par le nombre de mots concernés. Chaque ensemble masque et test est alors précédé du numéro de référence du mot d'entrée.

Pour illustrer ceci, dans l'exemple précédent, supposons trois mots d'entrée. Les variables *a* et *b* sont placées en tête du premier mot, *c* en tête du troisième mot ; le second mot est indifférent. Le traitement du monôme *ac* peut se faire à partir des tables de données de la figure I.8 qui correspondent à chacune des solutions.

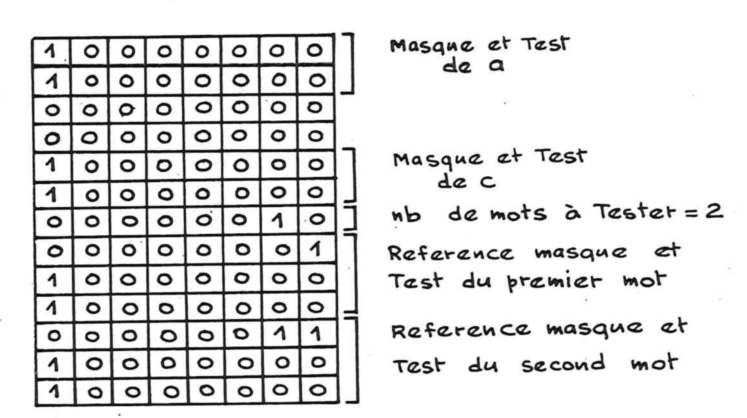


Fig. I.8

Le choix d'une solution doit être fait en tenant compte du nombre de mots d'entrée.

Le gros défaut de cette méthode est lié au traitement des fonctions simultanées ; celles-ci doivent être placées dans des tables successives. Il est évidemment possible d'associer cette idée de masquage à la méthode de l'organigramme, pour obtenir un traitement sur mot et une structure «orientée programme».

1.2.4. Méthode par adressage

Cette méthode considère que la combinaison des n variables d'entrée constitue l'adresse d'une valeur en mémoire. Il faut donc 2^n cases mémoires. Comme les mémoires sont organisées par mots de m bits, il est possible de traiter m fonctions simultanées.

Par exemple 7 fonctions de 5 variables nécessitent 32 mots de 7 bits. Il est bien sûr possible de minimiser le nombre de mots de l'espace mémoire occupé en décomposant les n variables d'entrées en deux groupes, un pour la sélection du mot, l'autre pour la sélection du (ou des) bit (s) dans le mot. Ainsi une seule fonction de huit variables peut se faire à partir de 32 mots de huit bits. Dans ce sens, notre exemple initial (I.1) est trivial car il suffit d'un mot de huit bits.

La structure de la solution est «orientée donnée», le programme de gestion doit simplement considérer les variations de longueur pour la sélection du mot et la sélection du ou des bits.

Un avantage important de la méthode est la longueur fixe de la structure en mémoire pour un nombre de variables donné. La mise en place est facile et immédiate, mais peut être longue car énumérative, et importante en espace mémoire.

1.2.5. Méthode par simulation

Le but de cette procédure est de délivrer l'utilisateur de la tâche de transcription dans une des structures précédentes. Il est en effet possible d'écrire une fonction sous une certaine forme proche de celle de l'utilisateur, de l'introduire en mémoire élément par élément (caractère codé ASCII en général) et de la traiter par un programme d'évaluation qui interprète la suite d'éléments considérée comme une table de données.

Soit par exemple une forme acceptant les variables, les variables complémentées, les opérateurs ET, OU et les parenthèses.

La méthode d'évaluation consiste à recopier l'expression dans une table de travail avec toujours un élément par case mémoire mais en remplaçant les variables par leur valeur mesurée. Le programme effectue alors des simplifications successives dans l'ordre : produit élémentaire, somme élémentaire, suppression des parenthèses devenues inutiles. Ce traitement est itéré jusqu'à obtention d'une seule valeur, celle de la fonction.

Il est de plus possible d'adjoindre au traitement une analyse syntaxique lors de l'introduction des données afin de tester la validité de l'expression.

Pour illustrer la procédure, considérons la fonction initiale sous sa forme (I.3) et supposons que

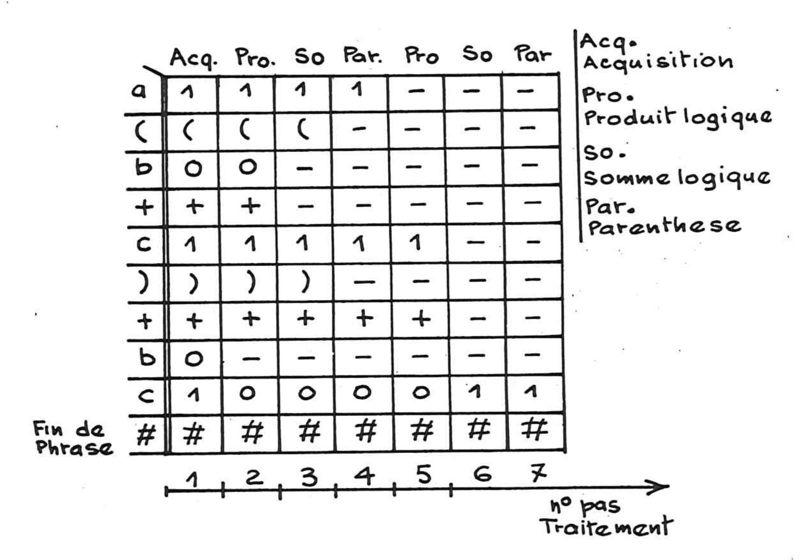
$$a = 1, \ b = 0, \ c = 1.$$

La figure I.9 représente la table de données et les différentes valeurs de celle-ci au cours du traitement.

L'expression est recopiée et au fur et à mesure des condensations, on fait figurer des espaces.

Les tests des ET consistent à détecter une chaine de 0 ou de 1 et d'en faire la synthèse.

Pour le OU, on cherche l'intérieur des parenthèses les plus intérieures. Les parenthèses à supprimer doivent entourer une valeur 0 ou 1.



	Acq.	Pro.	So	Par.	Pro	So	Par
a	1	1	1	1	–	–	–
(	(	(	(	–	–	–	–
b	0	0	–	–	–	–	–
+	+	+	–	–	–	–	–
c	1	1	1	1	1	–	–
)	)	)	)	–	–	–	–
+	+	+	+	+	+	–	–
b	0	–	–	–	–	–	–
c	1	0	0	0	0	1	1
#	#	#	#	#	#	#	#

Fig. I.9

De façon analogue, il est possible d'interpréter diverses formes d'écriture et ceci sur machine universelle. Toutefois, le traitement est en général assez long ; on est donc limité à des tables relativement courtes, c'est-à-dire à un petit ensemble de fonctions.

Pour conserver le grand intérêt d'une écriture proche de celle des utilisateurs, tout en ayant un temps de traitement compatible avec les impératifs industriels, sont apparus sur le marché les automates programmables.

I.3. Les langages spécifiques pour la mise en œuvre des fonctions logiques

1.3.1. L'automate programmable

Les automates ou contrôleurs programmables sont des machines dans lesquelles l'interpréteur précédant est en général câblé ou microprogrammé. La procédure de traitement est basée sur la scrutation cyclique de la mémoire de programme. Nous ne prendrons pas en compte dans ce

paragraphe certaines machines de haut de gamme, véritables machines universelles à langage orienté logique disposant soit d'un interpréteur rapide soit d'un compilateur.

La structure générale d'un automate programmable est donnée par la figure I.10.

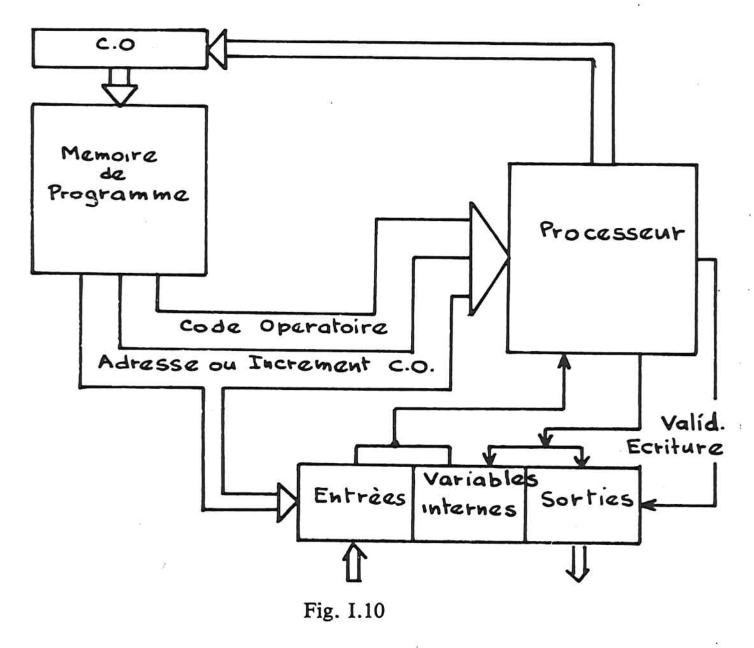


Fig. I.10

Le compteur ordinal CO pilote la mémoire de programme qui fournit une instruction. Celle-ci comporte deux parties, l'une pour le code opératoire, l'autre pour l'adresse des entrées, sorties, variables internes. Le déroulement normal se fait par simple incrémentation de CO, toutefois lors de saut programmé, la partie adresse de l'instruction sert d'incrément. Les sauts sont donc toujours vers l'avant, c'est-à-dire vers une adresse plus élevée.

Dans le cycle mémoire, il convient d'intégrer l'introduction des entrées et l'application des sorties. En dénotant *E* une acquisition d'une ou des entrées, *S* une affectation d'une ou des sorties et *T* un ensemble de traitements, il apparaît alors trois grandes classes de dispositifs représentées par les diagrammes de la figure I.11 ; les autres possibilités s'y ramènent soit au niveau du cycle, soit au niveau des sous-cycles éventuels.

Le cas (a) correspond à un mode strictement synchrone vis-à-vis des entrées-sorties. La seule variante possible concerne la fixité du temps de cycle. En effet, sur certaines machines, les sauts programmés raccourcissent le temps d'exploration de la mémoire. Ce mode de réalisation permet d'éviter la majorité des aléas.

Dans le mode (b), on conserve la scrutation globale des entrées en début de cycle, mais on applique les sorties au fur et à mesure de leur définition dans le programme. L'ordre dans lequel elles apparaissent peut alors présenter une importance sur le plan des aléas de continuité dans les commandes (chevauchement de sorties contradictoires ou discontinuité dans les actions).

a) → E → T → S

b) → E → T → S → T → .. T → S

c) → E → T → S → E → T → S → .. . S

Fig. I.11

Le cas (c) examine les entrées à chaque apparition de celles-ci. Si, dans certains cas, cette procédure permet pour certains traitements urgents de prendre en compte plusieurs fois la même instruction ou le même programme au cours d'un cycle, elle reste d'un maniement fort dangereux et il devient impératif de figer dans une variable interne la valeur d'une entrée intervenant en plusieurs endroits dans une équation ou un ensemble d'équations.

Le jeu d'instructions d'un automate programmable peut se décomposer en trois grandes classes : tout d'abord les instructions servant à la programmation booléenne classique, puis celles orientées vers le traitement des automatismes séquentiels industriels, et qui comprennent des mises à zéro ou à un de variable interne ou de sortie, des branchements, des pas à pas, des comptages et des temporisations ; enfin on trouve les extensions numériques de calcul sur mot. Nous reviendrons ultérieurement sur ces deux dernières catégories.

1.3.2. Les primitives booléennes

Une partie de la programmation d'automatismes logiques revient à décrire dans un langage approprié, un ensemble d'équations booléennes. Cette description se réalise, suivant le type de processeur, soit en considérant les éléments logiques traditionnels, soit en transposant des structures informatiques. Nous pouvons ainsi dégager six grandes classes de primitives :

— schéma à relais (ladder diagram)

— logigramme
— analyseur booléen
— unité logique
— notation polonaise inverse
— organigramme

I.3.2.1. Echelle à relais

Les automates programmables étant surtout destinés, initialement, à remplacer les armoires de relayage, un grand nombre de machines ont été réalisées pour permettre à un utilisateur familiarisé avec les diagrammes à relais (ladder diagram) de faire facilement l'implantation de son câblage. Il convient toutefois de noter qu'une transcription directe est illusoire et que, pour éviter des aléas, il est nécessaire de tenir compte de la nature séquentielle du traitement.

Cette écriture possède toujours une grande vogue, principalement aux U.S.A., les instructions suivent alors les éléments du schéma à relais, soit :

Adresse — contact ouvert au repos et désigné par son adresse.

Adresse — contact fermé au repos et désigné par son adresse.

ouverture d'une branche.

fermeture d'une branche.

Adresse — assignation à une sortie, désignée par l'adresse, de la valeur obtenue pour l'ensemble des branches qui lui sont connectées.

Adresse — même chose que mais avec le complément de la valeur.

Le schéma à relais de la figure I.12 pourra dans cet esprit se traduire par la suite d'instructions qui lui est juxtaposée. Cet exemple correspond à notre fonction initiale sous sa forme (I.2). A la place des numéros correspondant à l'adresse des variables, nous avons mis les variables elles-mêmes.

Il est à noter que, suivant les constructeurs, la notion d'ouverture et de fermeture de branche peut varier. Certains admettent en effet des ouvertures en cascade, mais ajoutent alors une instruction spéciale de retour à la dernière ouverture effectuée. Certains schémas, comme celui de

la figure I.13, qui comportent des liaisons pontées entre branches, ne peuvent être mis en œuvre qu'après transformation, la structure la plus traditionnelle restant la forme série-parallèle.

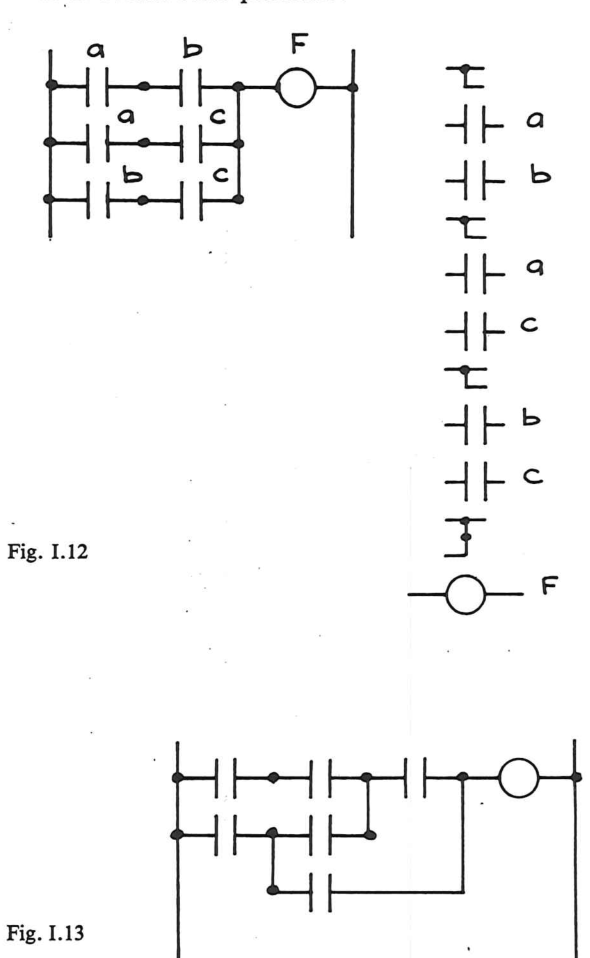


Fig. I.12

Fig. I.13

I.3.2.2. Le logigramme

Cette écriture permet de transcrire des schémas logiques correspondant à des équations réalisées à partir d'un certain nombre de modules logiques prédéfinis dont la sortie porte le numéro d'une sortie ou d'une variable interne. La description se fait en donnant successivement les entrées du module, puis en indiquant sa fonction et son numéro de sortie.

Considérons à titre d'exemple le jeu d'instructions suivant :

— CH	adresse	chargement d'une entrée de module avec la valeur de la variable adressée.
— CHC	adresse	chargement d'une entrée de module avec le complément de la valeur de la variable adressée.
— ET	adresse	module ET dont la sortie est transmise à l'adresse indiquée.
— OU	adresse	module OU dont la sortie est transmise à l'adresse indiquée.

Son application à la fonction (I.3) peut être concrétisée sur la figure I.14.

Dans certains cas, on peut mettre immédiatement ET VI3 après OU VI1 sans redéfinir les entrées puisqu'elles sont identiques.

Il est possible de trouver des modules plus complexes (NAND, NI, OU exclusif) et même composites comme celui de la figure I.15.

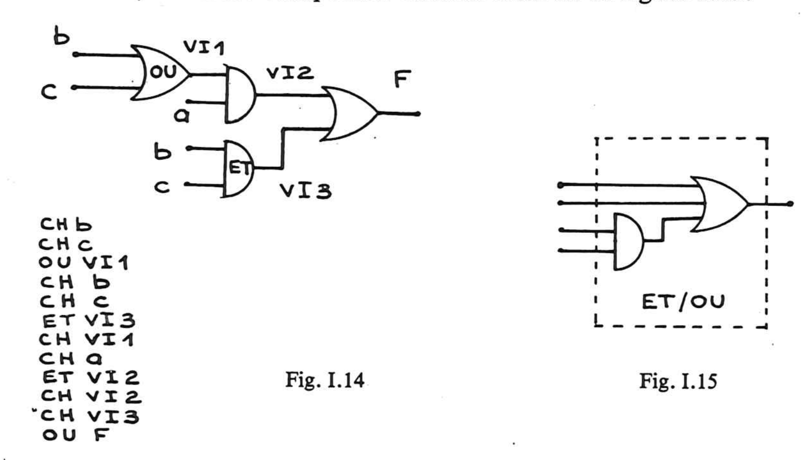


Fig. I.14

Fig. I.15

I.3.2.3. L'analyseur booléen

Avec ce type de processeur, l'équation est implantée en mémoire élément par élément, pratiquement comme elle s'écrit sur le papier. Toutefois, l'affectation du résultat (opérateur =) se place en fin d'équation. D'autre part, le nombre de niveaux de parenthèses est souvent limité à 1 ou même à 0. Il est à noter que la possibilité d'ouvrir plusieurs parenthèses mais avec une fermeture globale unique correspond en fait à un seul niveau.

Le processeur, automate synchrone câblé, traite l'équation élément par élément et établit son évaluation en tenant compte du fait qu'une

variable à «1» dans une somme, fixe le résultat à «1» tandis qu'un zéro dans un produit le rend égal à zéro. L'automate analyseur tient compte des résultats partiels figurant à l'intérieur des parenthèses.

I.3.2.4. Unité logique

Sur certaines machines, le processeur suit les mêmes principes que l'unité de traitement des calculateurs universels. Il est bâti autour d'un bloc logique et d'un accumulateur. Pour éviter d'avoir à utiliser trop souvent des variables internes pour mémoriser les résultats intermédiaires, on trouve parfois des registres internes organisés sour forme de pile. Ceci rend possible les opérations entre l'accumulateur et le sommet de la pile, le rangement dans la pile...

Les instructions de base sont :

— CH	adresse	chargement dans l'accumulateur de la valeur de la variable adressée.
— CHC	adresse	chargement dans l'accumulateur du complément de la valeur de la variable adressée.
— RA	adresse	rangement du continu de l'accumulateur à l'adresse indiquée. Le contenu n'est pas affecté.
— RAC	adresse	rangement du complément du contenu de l'accumulateur à l'adresse indiquée sans affecter le contenu.
— OU — ET	adresse adresse	opération entre le contenu de l'accumulateur et une variable adressée. Le résultat est mis dans l'accumulateur.

Pour la fonction (I.3), le programme correspondant pourrait s'écrire à partir de la position 0 en mémoire :

0	CH b	4	CH b
1	OU c	5	ET c
2	ET a	6	OR VI
3	RA VI	7	RA F

I.3.2.5. Notation polonaise inverse

Cette écriture permet de supprimer les problèmes de parenthèses. Elle consiste à écrire d'abord les variables intervenant dans un calcul, puis les opérateurs. Le processeur est organisé autour d'une mémoire organisée en structure de pile du type LIFO (stack machine). Les primitives se réduisent simplement à l'appel d'une variable ou de son complément (*CH ou CHC*) qui charge le sommet de la pile, à l'opérateur PAS qui complémente le

sommet de la pile, aux opérateurs ET, OU, XOU (ou exclusif) qui prennent comme opérandes les deux éléments au sommet de la pile et qui place le résultat au sommet de la pile, et enfin au rangement (RA), affectation à une variable de la valeur du sommet de la pile.

La fonction (I.3) en notation polonaise inverse devient :

$$abc + .bc. + = F$$

On en déduit le programme à partir de la position 0 de la mémoire :

0 CH a	4 ET	8 OU
1 CH b	5 CH b	9 RA F
2 CH c	6 CH c	
3 OU	7 ET	

I.3.2.6. Organigramme

Le processeur qui correspond à ce mode de traitement doit nécessairement disposer d'une instruction de saut. Mais pour conserver l'aspect de scrutation cyclique de la mémoire, les sauts ne peuvent se faire que vers l'avant, c'est-à-dire par incrémentation du compteur ordinal. Les sauts sont conditionnés à la valeur d'un registre d'un bit chargé par appel d'une variable ou par appels successifs d'une liste de variables reliées par un ET implicite. Le saut peut être absolu ou relatif. Si la condition est à «0», la progression se fait alors par simple incrémentation (+1) du compteur ordinal.

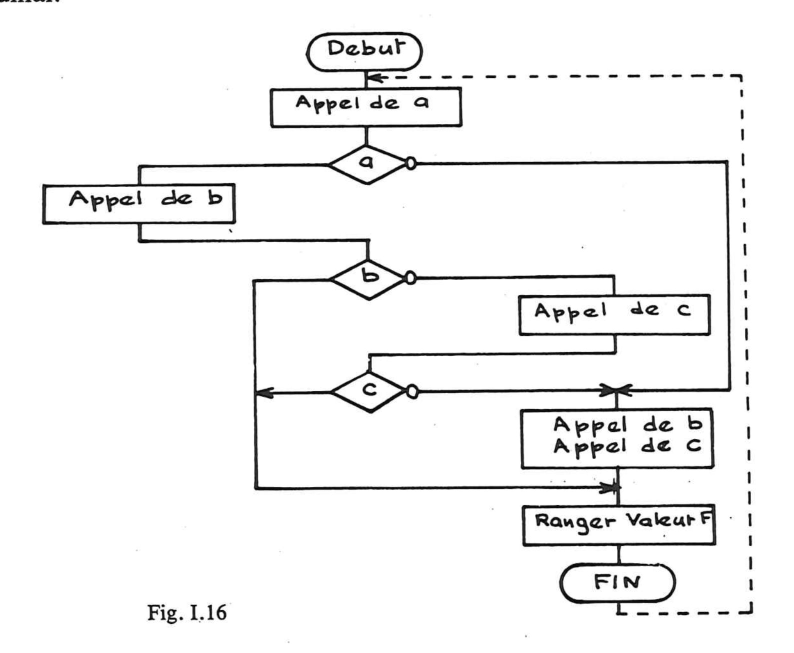


Fig. I.16

Le registre interne est initialisé à 1 en début de cycle et remis à 1 après un saut effectivement réalisé.

Avec le jeu d'instructions suivant :

— CH adresse : chargement du registre interne par la valeur de la variable adressée. Des chargements en cascade forment un produit logique.

— CHC adresse : chargement du registre interne par le complément de la variable adressée.

— RA adresse : rangement du résultat.

— SI «O» ALLER A + n où *n* est la valeur de l'incrément.

— SI «1» ALLER A + n

L'organigramme de la figure I.16 se traduit par le programme implanté à partir de l'adresse 0 en mémoire.

0 CH *a*
1 SI «0» ALLER A + 5
2 CH *b*
3 SI «1» ALLER A + 5
4 CH *c*
5 SI «1» ALLER A + 3
6 CH *b*
7 CH *c*
8 RA F

I.4. Notion de performance

Un circuit combinatoire est fait pour donner la correspondance entre une combinaison de sortie et une combinaison d'entrée, mais cela ne peut se faire instantanément. Il y a un temps de réponse. Dans les systèmes programmés, il est lié au temps d'exécution du programme. Il est donc très difficilement chiffrable pour une machine universelle, par contre pour un automate programmable, il est directement lié au temps de cycle. Ce temps est en général de quelques millisecondes : il dépend en fait de la capacité mémoire.

Ces systèmes n'ont donc d'intérêt que pour la logique industrielle classique, et ne constituent pas une solution à tous les problèmes logiques. Toutefois, il est un point très important. Lorsqu'ils sont utilisables, ils permettent de mettre en quelques boîtiers ce qui auparavant aurait nécessité un nombre important de modules. De plus, un même ensemble est facilement réutilisable par simple changement du programme ; les corrections, extensions, modifications sont donc énormément facilitées.

I.5. Les réalisations spéciales

Il est possible de matérialiser industriellement de manière économique, les fonctions combinatoires en utilisant des assemblages de réseaux logiques particuliers ou de circuits de mémorisation. Ces réalisations spéciales sont obtenues soit par implantation directe en logique câblée, éventuellement pré-programmée avant la mise en service, soit par la construction de structure séquentielle appropriée et bien adaptée, permettant un traitement programmé.

I.5.1. Implantation directe

Elle est basée sur l'utilisation directe de circuits multiplexeurs, ou de réseaux logiques programmables (PLA, PAL, FPLA) ou de circuits de mémorisation (PROM, ROM, RAM).

I.5.1.1. Circuits multiplexeurs

Un multiplexeur à une variable de sélection a (figure I.17) est un système logique matérialisant la fonction combinatoire.

$$S = \bar{a}.x + ay$$

En appliquant de manière itérative la relation :

$$S = F(x_1, x_2, \text{--}x_{i-1}, x_i, x_{i+1}, \dots x_n) =$$

$\bar{x}_i.F(x_1, x_2 \dots x_{i-1}, 0, x_{i+1} \dots x_n) +$

$x_i.F(x_1, x_2 \dots x_{i-1}, 1, x_{i+1}, \dots x_n).$

qui permet de transformer, à l'aide d'un multiplexeur à une variable de sélection x_i (fig. I.18) le problème de la synthèse d'une fonction combinatoire de n variables en un problème de synthèse de 2 fonctions combinatoires de n-1 variables, il est possible d'obtenir un assemblage de multiplexeurs matérialisant une fonction combinatoire donnée.

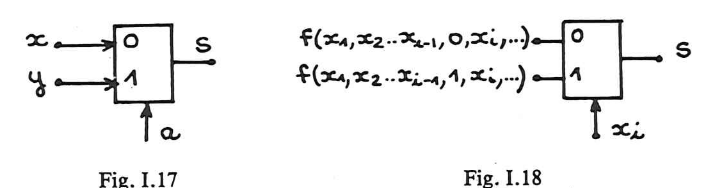


Fig. I.17 Fig. I.18

En utilisant cette méthode, la fonction F définie par la table de vérité de la figure I.1, se représente en développant successivement par rapport aux variables a, puis b, puis c, par la structure arborescente donnée par la figure I.19.

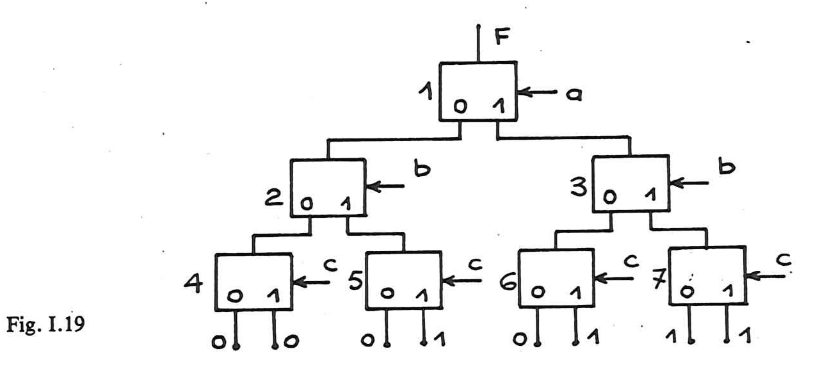


Fig. I.19

La comparaison de cette structure et de l'arbre de décision logique de description de la même fonction F donné par la figure I.20, montre l'équivalence des deux descriptions, un multiplexeur à une variable de sélection x_i remplaçant un élément de test d'une même variable x_i.
Par application directe de cette remarque et de la procédure de regroupement détaillée en I.2.1., on obtient les réalisations représentées par la figure I.20, analogue à la figure I.4 et par la figure I.21.

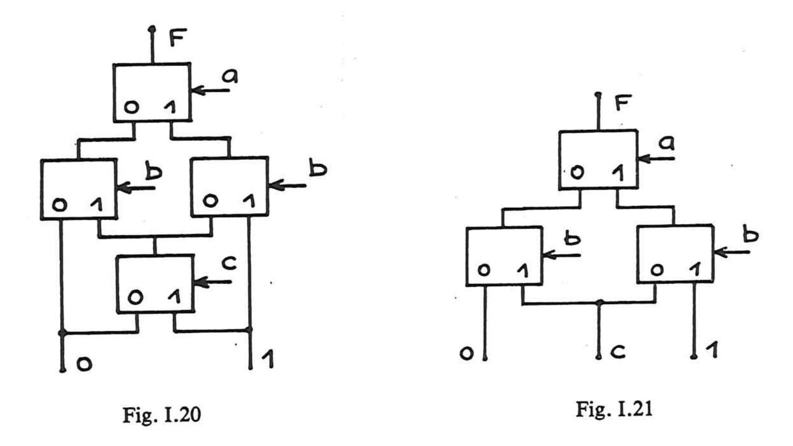


Fig. I.20 Fig. I.21

L'utilisation de multiplexeurs à plusieurs variables de sélection supprime les opérations de câblage. La fonction F définie en I.1 est matérialisée par un multiplexeur à 3 variables de sélection a, b, c (figure I.22).

$$F = S = \bar{a}\bar{b}\bar{c}x_0 + \bar{a}\bar{b}cx_1 + \bar{a}b\bar{c}x_2 + \bar{a}bcx_3 + a\bar{b}\bar{c}x_4 + a\bar{b}cx_5 + ab\bar{c}x_6 + abcx_7$$

la colonne des valeurs imposées aux entrées annexes x_0, x_1, x_2.......x_7 correspondant à la colonne de définition de F dans la table de vérité.

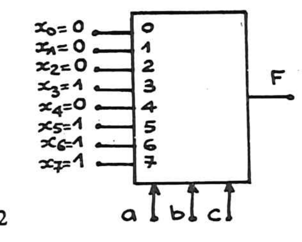


Fig. I. 22

I.5.1.2. Les réseaux logiques programmables et les circuits de mémorisation

Sans tenir compte des dispositifs annexes tels que circuits décodeurs, commande du type 3-états des sorties, ensemble de portes d'entrée programmables, permettant des extensions intéressantes des fonctions combinatoires implantées, les réseaux logiques programmables (PLA, FPLA, PAL) et les circuits de mémorisation (PROM, ROM, RAM) permettent la réalisation de fonctions booléennes représentées sous la forme de somme de produits des variables d'entrée et de leurs compléments.

Ces circuits à haute intégration sont soit programmés à la fabrication (ROM), soit programmables par l'utilisateur (PROM, RAM, FPLA, PAL). Leur classement peut être effectué en examinant les parties figées ou programmables des sommes de produits. Il est résumé par le tableau de la figure I.23 qui sera illustré et commenté sur un exemple simple, utilisant

Reseau ET \ Reseau OU	Fixé	Programmable
Fixé		P.R.O.M R.O.M R.A.M
Prog.	P.A.L	P.L.A F.P.L.A

Fig. I.23

le formalisme représenté par la figure I.24 qui permet d'alléger la représentation des réseaux ou des circuits de mémorisation. Sur cette figure, l'absence de croix à l'intersection d'une ligne et d'une colonne implique une liaison coupée, ou non établie, entre cette ligne et cette colonne. La présence d'une croix placée à l'intersection d'une ligne et d'une colonne dénote un court-circuit existant ou établi entre cette ligne et cette colonne. En utilisant cette convention, il est possible de détailler la structure interne du circuit à 4 entrées-4 sorties donné par la figure I.25 par le schéma de la figure I.26.

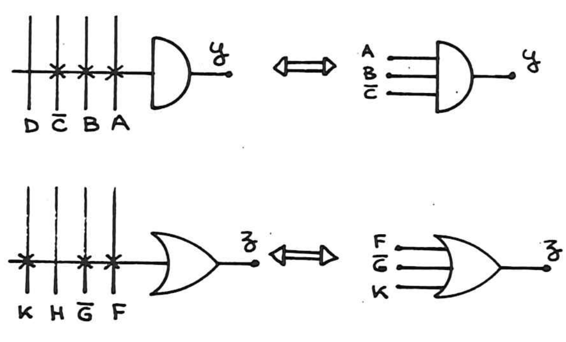


Fig. I.24

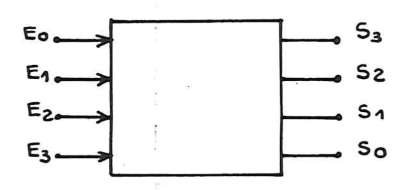


Fig. I.25

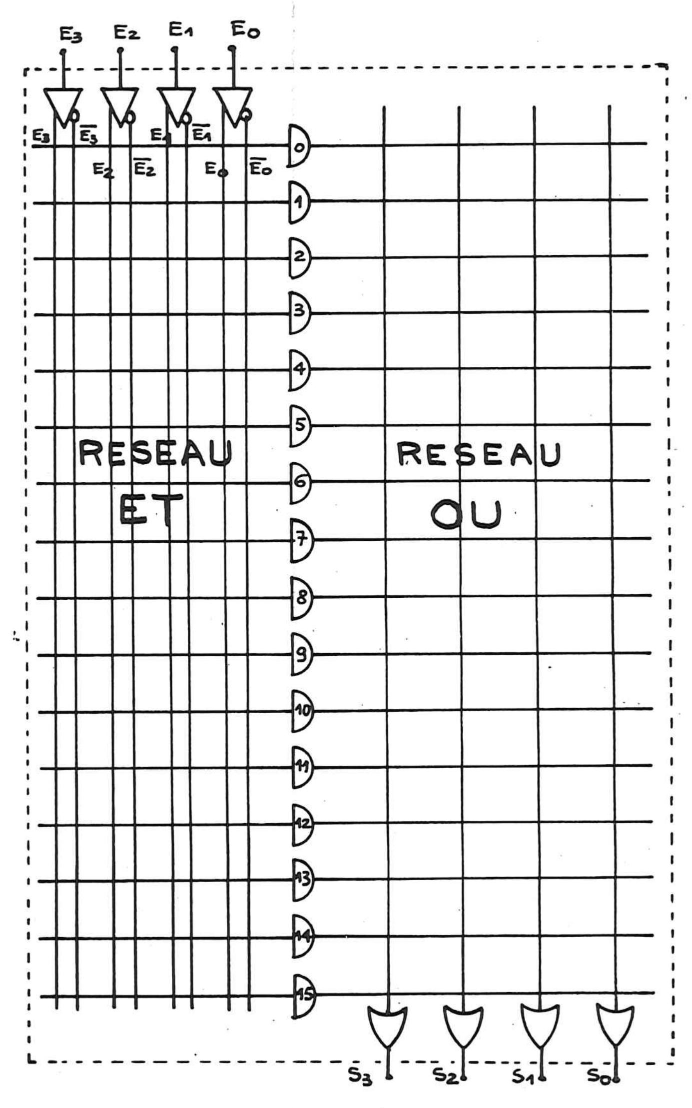


Fig. I.26

Ce circuit comporte un réseau de 16 portes ET permettant suivant le placement de croix aux intersections des lignes et des colonnes, d'obtenir divers produits des grandeurs d'entrée ou de leur complément, et un réseau de 4 portes OU permettant suivant la position des croix à l'intersection des lignes et des colonnes, d'obtenir 4 sommes quelconques des produits précédemment constitués.

Soit à matérialiser l'ensemble des fonctions simultanées des 4 variables a, b, c, d suivantes :

$$F_0 = ab + ac + bc$$

$$F_1 = a\bar{d} + b + \bar{a}\bar{c}$$

$$F_2 = \bar{a}\bar{b} + cd$$

$$F_3 = abc + \bar{b}\bar{d}$$

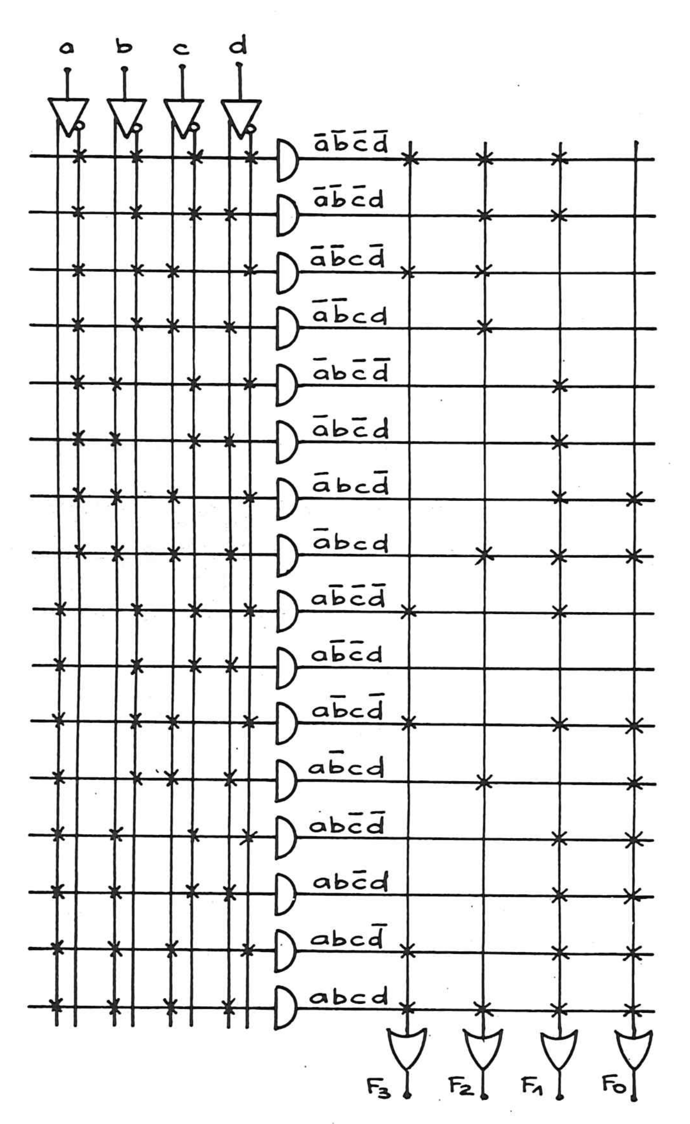


Fig. I.27

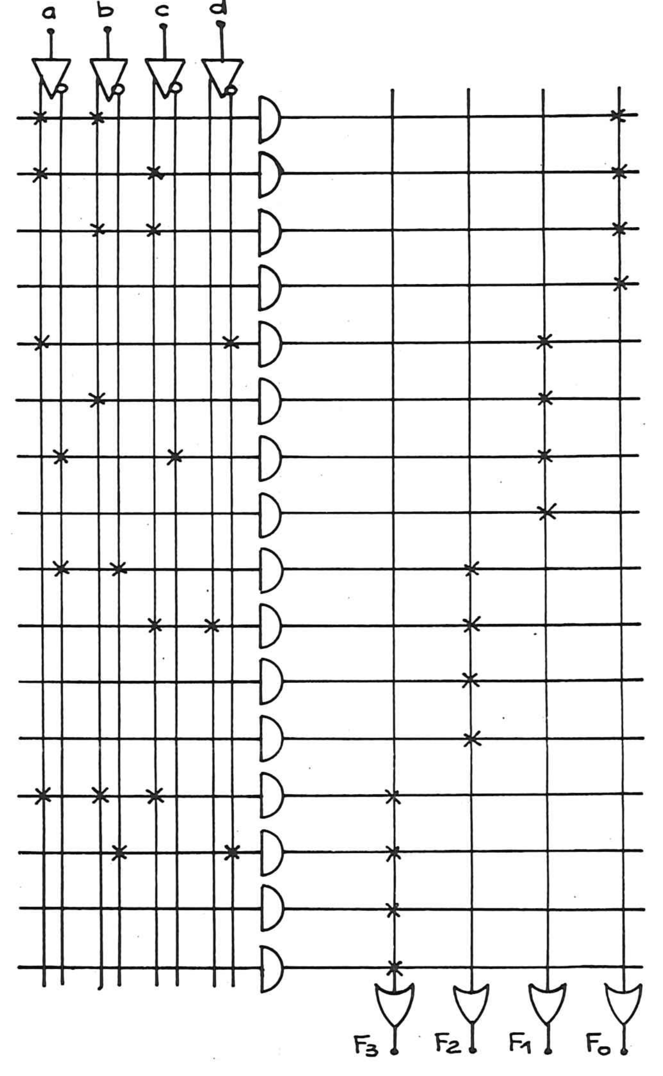


Fig. I.28

Dans le cas de l'implantation à l'aide de circuits de mémorisation (RAM, PROM, ROM), le réseau ET de la structure générale donnée par la figure I.26 est fixé comme le montre la figure I.27. Les 16 portes ET 0, 1, 2... 15 constituent le décodeur du mot d'entrée a b c d dénoté adresse.

L'ensemble des fonctions combinatoires s'obtient par simple sommation de termes élémentaires.

Dans le cas de l'implantation à l'aide de réseaux logiques programmables du type PAL (programmable array logic), le réseau OU de la structure générale donnée par la figure I.26 est fixé par exemple comme le montre la figure I.28. Il est alors nécessaire, les sorties étant affectées, de programmer judicieusement le réseau ET d'entrée.

Dans le cas de l'implantation à l'aide de réseaux logiques programmables du type PLA, FPLA, les réseaux OU et ET de la structure générale ne sont pas fixés à priori, mais programmables, ce qui augmente la liberté de manœuvre du concepteur, mais ne facilite pas nécessairement une programmation systématique.

Dans certaines applications, il est nécessaire de disposer d'un nombre d'entrées ou de sorties ou de termes produits plus importants. Ces extensions s'obtiennent par agencement de plusieurs réseaux, par l'utilisation d'une commande CE de sélection (chip enable), les circuits possédant des sorties du type trois états, et d'un décodeur, ou d'ensemble de portes programmables (Field programmable gate array) du type NON-ET, ET, NON-OU, OU. Les figures I.29 et I.30 correspondent à une extension des entrées, la figure I.31 à une augmentation du nombre des produits, et la figure I.32 à une extension du nombre de sorties.

L'intersection vierge non utilisée d'une colonne et d'une ligne correspond initialement, suivant le type de circuit (ROM, PROM, PLA...) et la technologie, soit à une liaison établie soit à une liaison coupée.

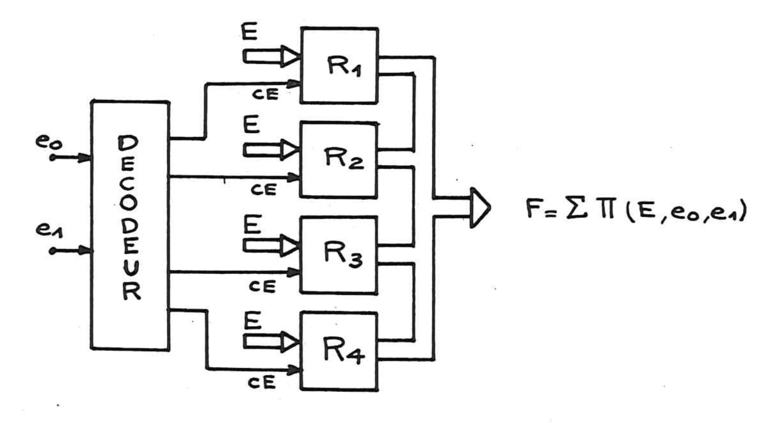


Fig. I.29

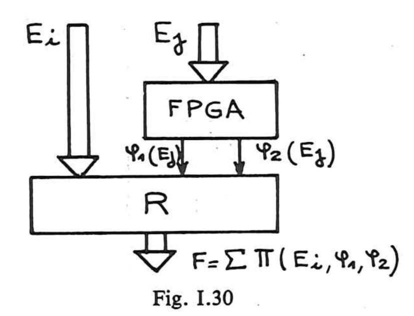


Fig. I.30

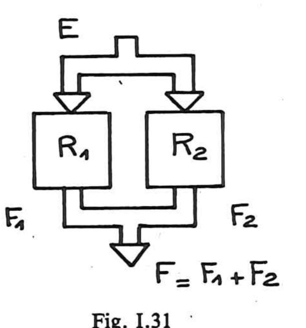


Fig. I.31

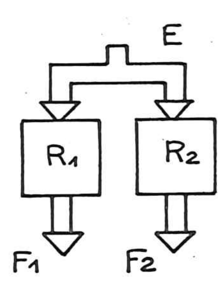


Fig. I.32

1.5.2. Construction d'une structure séquentielle

La construction à l'aide de réseaux logiques ou de circuits de mémorisation d'un processeur permettant un traitement séquentiel peut s'effectuer de différentes manières suivant le mode d'adressage retenu et le jeu d'instructions de la machine élémentaire élaborée.

Nous nous limitons dans ce chapitre à la description sur un exemple déjà traité des différentes étapes d'élaboration de la structure dans le cas de l'utilisation d'un circuit de mémorisation du type PROM, d'une structure à 2 adresses et d'un jeu de 2 instructions élémentaires

Ces étapes sont :

— Obtention d'un arbre de décision logique
— Représentation tabulaire de l'arbre
— Codage du tableau
— Implantation de la machine séquentielle.

Soit à matérialiser une structure séquentielle de traitement des fonctions simultanées :

$$F_1 = a\bar{d} + b + \bar{a}\bar{c}$$

$$F_2 = \bar{a}\bar{b} + cd$$

— *Obtention d'un arbre de décision logique*

En regroupant les affectations identiques des sorties et en numérotant

les différents éléments de l'arbre de décision binaire de la figure I.5a), l'on obtient la figure I.33.

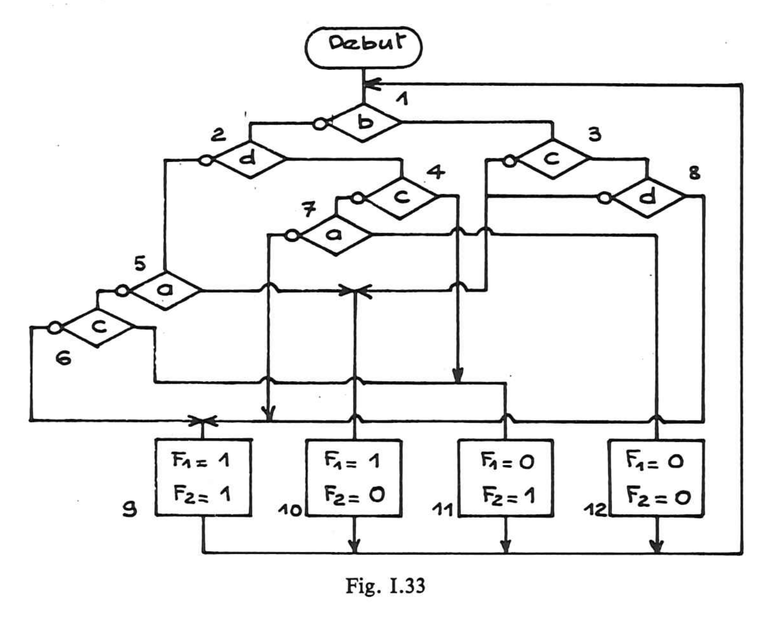


Fig. I.33

— *Représentation tabulaire de l'arbre*

En dénotant instruction de test un élément de test, instruction de sortie un élément d'assignation de valeurs aux fonctions *F1* et *F2* et en utilisant la numérotation des éléments, il est possible de représenter l'arbre de décision logique par le tableau équivalent donné par la figure I.34.

		Instruction Test			Instruction Sortie		
			inst. suivante			valeurs S	
n° instruc.	nature instruc	varia. Testée	Test 1	Test 0	no instruc suiv.	F1	F2
1	Test	b	3	2			
2	Test	d	4	5			
3	Test	c	8	10			
4	Test	c	11	7			
5	Test	a	10	6			
6	Test	c	11	9			
7	Test	a	12	9			
8	Test	d	9	10			
9	Sortie				1	1	1
10	Sortie				1	1	0
11	Sortie				1	0	1
12	Sortie				1	0	0

Fig. I.34

— *Codage du tableau*

Il découle de la nécessité d'une mémorisation dans la PROM, et implique le choix d'un code de représentation de la nature de l'instruction (α), du numéro de l'instruction ($E_0\ E_1\ E_2\ E_3$) et de la variable testée (xy). En choisissant les correspondances définies par les tableaux de la figure I.35 et en translatant les informations relatives aux instructions de sortie dans les zones hachurées inutilisées de l'instruction suivante dans le cas d'un test, on obtient le tableau codé de la figure I.36.

Inst.	α
Sortie	0
Test	1

	Variable Testée			
	a	b	c	d
x	0	1	0	1
y	0	0	1	1

	Numero Instruction											
	1	2	3	4	5	6	7	8	9	10	11	12
E_0	0	1	0	1	0	1	0	1	0	1	0	1
E_1	0	0	1	1	0	0	1	1	0	0	1	1
E_2	0	0	0	0	1	1	1	1	0	0	0	0
E_3	0	0	0	0	0	0	0	0	1	1	1	1

Fig. I.35

				inst	varia Test		Test 1				Test 0			
E_3	E_2	E_1	E_0	α	y	x	E'_3	E'_2	E'_1	E'_0	E'_3	E'_2	E'_1	E'_0
0	0	0	0	1	0	1	0	0	1	0	0	0	0	1
0	0	0	1	1	1	1	0	0	1	1	0	1	0	0
0	0	1	0	1	1	0	0	1	1	1	1	0	0	1
0	0	1	1	1	1	0	1	0	1	0	0	1	1	0
0	1	0	0	1	0	0	1	0	0	1	0	1	0	1
0	1	0	1	1	1	0	1	0	1	0	1	0	0	0
0	1	1	0	1	0	0	1	0	1	1	1	0	0	0
0	1	1	1	1	1	1	1	0	0	0	1	0	0	1
1	0	0	0	0	–	–	0	0	0	0	–	–	1	1
1	0	0	1	0	–	–	0	0	0	0	–	–	1	0
1	0	1	0	0	–	–	0	0	0	0	–	–	0	1
1	0	1	1	0	–	–	0	0	0	0	–	–	0	0
							E'_3	E'_2	E'_1	E'_0			F_1	F_2
							INST. SUIV.				SORTIE			

Fig. I.36

— *Implantation de la machine séquentielle*

Une implantation possible de la machine séquentielle est donnée par la figure I.37. Elle comprend :

— une mémoire PROM à 4 entrées E0, E1, E2, E3 et 11 sorties, dont le réseau OU a été particularisé afin de représenter la partie entourée en traits forts de la table de la figure I.32.

— un ensemble de 4 bascules D0, D1, D2, D3 mémorisant l'adresse E0, E1, E2, E3 de l'instruction en cours. Ces bascules constituent le compteur ordinal et sont initialisables à l'adresse 0, 0, 0, 0 correspondant à l'instruction 1.

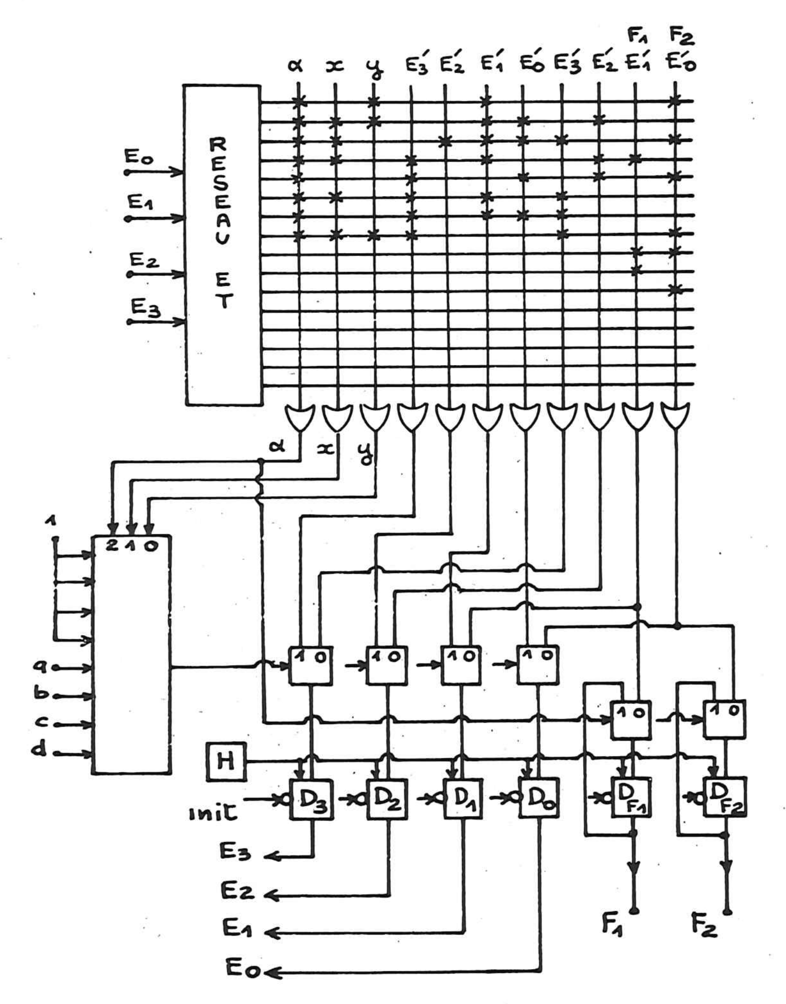


Fig. I.37

— un ensemble de 2 bascules DF1, DF2 de sortie.

— un ensemble composé d'un multiplexeur à 3 variables de sélection, nature de l'instruction, variable testée et de 6 multiplexeurs élémentaires de commande des bascules.

Le rôle de ce système est de permettre suivant la nature de l'instruction en cours un aiguillage correct vers le compteur ordinal de l'adresse de l'instruction suivante, et l'affectation ou le maintien des valeurs de sortie à l'entrée des bascules DF1 et DF2.

III - LES MÉTHODES D'IMPLANTATION DES OUTILS DE DESCRIPTION

Les outils de description : grafcet, réseau de Pétri définis dans le chapitre présentent industriellement un grand intérêt. Toutefois la représentation par un graphe d'une structure séquentielle ne constitue pas, sur le plan de la synthèse, une fin en soi. Il importe de pouvoir faire d'une manière générale, le passage à la mise en œuvre sur un dispositif technologique quelconque, et plus particulièrement compte tenu de nos objectifs, d'utiliser les systèmes programmés de toute nature. Ce chapitre va tenter de dégager les méthodes d'implantation en ne prenant en considération que les traits technologiques fondamentaux. Les réalisations feront l'objet du chapitre suivant.

III.1. Synchronisme - Asynchronisme

Les systèmes programmés : calculateurs universels, structures spécialisées, automates programmables... sont pilotés par une horloge principale et constituent donc par essence des machines synchrones. Ce premier niveau de synchronisme ne présente toutefois pas un intérêt considérable pour notre étude car il est nécessaire de préciser au niveau des acquisitions des entrées, des affectations de sorties et du traitement les notions de synchronisme et d'asynchronisme. Dans cette optique, il est possible de dégager un second niveau de synchronisme qui est relatif aux entrées-sorties.

La prise en compte des entrées est synchrone lorsque toutes les valeurs des entrées sont figées avant de commencer le traitement. L'application des sorties peut aussi être synchrone si celle-ci se fait globalement à la fin de tous les traitements.

Une telle définition nécessite dans la pratique la création d'une zone mémoire, image des entrées et des sorties, à laquelle on adjoint une zone mémoire des variables internes.

En début de calcul, on recopie automatiquement les valeurs effectives des entrées dans la zone image correspondante. En fin du calcul (portant uniquement sur les zones images), on applique aux sorties réelles le contenu de la zone image élaboré durant le traitement.

Il existe également un troisième niveau de synchronisme trop souvent négligé ou ignoré dansla pratique. Il est relatif au calcul des conditions de franchissement des transitions et à l'évolution de l'activité ou du marquage. On peut avoir asynchronisme à ce niveau, même si le cycle de traitement des entrées-sorties est synchrone.

En prenant par exemple une description par grafcet, nous dirons que le traitement de l'évolution est synchrone s'il n'y a pas interaction entre évolution et calcul des conditions d'évolution en cours d'un même cycle de traitement. Dans la pratique le traitement commence donc par le calcul de l'ensemble des conditions d'évolution ou de celles susceptibles d'être vérifiées. Les conditions d'évolution étant calculées et figées, les nouvelles étapes actives et inactives sont alors établies, ce qui détermine l'évolution de l'activité du graphe.

III.2. Description par grafcet et synchronisme

Les conditions d'évolution de l'activité ou du marquage d'un grafcet comprennent 2 parties :

Règle de base

Le franchissement d'une transition entraîne l'activation de toutes les étapes de sortie de la transition, et la désactivation de toutes ses étapes d'entrée.

Règles de complément

RC1 : Plusieurs transitions simultanément franchissables sont simultanément franchies.

RC2 : Lorsque, par application de la règle de franchissement, une même étape doit être désactivée et activée, elle reste active.

Examinons l'impact pratique des règles de complément. A l'échelle des temps de traitement, il est possible que, durant un cycle, l'état du système se soit établi de façon à permettre le franchissement de plusieurs transitions. Cette situation peut se produire dans des grafcets décrivant des sous-processus indépendants ou comportant des branches à évolution parallèle et simultanée. La matérialisation programmée ne pose pas dans ce cas de difficulté. Elle est par contre plus délicate lorsque deux transitions franchissables possèdent une même place d'entrée comme dans l'exemple représenté par la figure III.I.

Fig. III.1

Dans cette situation si les événements EV1 et EV2 sont produits simultanément, les étapes 2 et 3 deviennent actives et l'étape 1 inactive, par application directe de la règle RC1. Pour qu'au niveau du traitement cela puisse être possible, il importe de considérer dans un premier temps les transitions à franchir (ici t1 et t2) puis dans un second temps de désactiver toutes les étapes d'entrée de cet ensemble de transition et d'activer toutes les étapes de sortie. Il faut donc faire un traitement synchrone ou parallèle. Un traitement asynchrone consisterait à considérer par exemple dans un premier temps la transition t1, de calculer sa condition d'évolution, puis, comme celle-ci est vérifiée, d'activer l'étape 2 et de désactiver l'étape 1. Lorsqu'on vient maintenant à traiter la transition t2, l'étape 1 étant devenu inactive, le franchissement est impossible. Le traitement asynchrone hiérarchise les conflits. Il devient alors évident qu'un tel traitement ne s'adapte pas à une description par grafcet nécessitant l'application de la règle de complément RC1.

Etendons un peu ce problème de conflit en considérant le grafcet de la figure III.2 et en supposant t1 et t2 franchissables simultanément. Après un traitement synchrone, on obtient les étapes 3 et 4 actives. Par contre, si on considère un traitement asynchrone, avec l'ordre d'examen des transitions t1 t3 t2, seule l'étape 5 est active, par contre, avec l'ordre t1 t2 t3, on retrouve le marquage du traitement synchrone. Cette remarque nous amène à penser que, formellement, il existe réellement deux modes de description permettant soit une matérialisation synchrone ou parallèle, soit une matérialisation asynchrone ou série. Il faut donc dès le départ et compte-tenu des caractéristiques du système programmé utilisé, effectuer un choix.

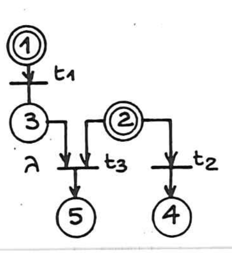


Fig. III.2

La règle RC2 sous-entend tout d'abord le fait qu'une étape activée reste activable. Pour un grafcet sans conflit, cela ne devrait normalement pas se produire. Cette même règle considère le cas de l'activation et la désactivation simultanée d'une étape. Une première possibilité de configuration entrant dans cette catégorie est donnée par la figure III.3. L'étape 2 est à la fois étape d'entrée et de sortie de la transition t1. Le grafcet comporte une boucle. On a alors souvent l'habitude de supprimer les arcs entre l'étape 2 et t1 en remplaçant l'événement EV associé à t1 par $EV.\overline{\varphi_2}$ où φ_2 représente l'activité de l'étape 2. Il convient de noter que cela n'a de sens que s'il n'y a pas de conflit entre t1 et t2 car, dans ce cas, la structure de la figure III-3 maintient le marquage de l'étape 2, alors qu'après la modification précédente cela n'est plus possible.

Fig. III.3

Une autre situation entraînant l'application de RC2 est telle qu'on puisse avoir simultanément le franchissement d'une transition d'entrée et d'une transition de sortie d'une même étape. Ceci implique que deux étapes qui se suivent soient actives simultanément et entraîne la réactivation ultérieure d'une étape déjà activée, sauf si l'événement associé à la transition t2 (fig. III.4) disparaît après le premier franchissement, ce qui se produit si on a un blocage par l'activité de l'étape 1 (notée $\varphi 1$), ou encore si les événements qui engendrent les franchissements simultanés sont fugitifs et n'existent que pour un cycle de traitement.

Fig. III.4

Considérons par exemple le cas élémentaire d'un cycle d'opérations qui évolue à chaque apparition d'un front montant d'une commande *a*. Le cycle comporte successivement les opérations *OP1*, puis *OP1* et *OP2* puis *OP2* et *OP3* et enfin *OP3*. Un grafcet pourrait être celui de la figure III-5.

Ici encore, avec la règle RC2, la possibilité de franchissement simultané exige un traitement synchrone, auquel nous devons ajouter la priorité de l'activation sur la désactivation. Finalement, compte-tenu de toutes ces remarques, nous définissons :

Grafcet de type 1 : c'est un grafcet sans boucle, sans conflit, où les

événements sont indépendants de l'activité des étapes, et tel que l'analyse des diverses possibilités de franchissement, prises une à une sur la structure non interprétée, ne révèle aucune réactivation d'étape. Ceci revient à attribuer à chaque transition un événement indépendant du processus à contrôler, et à examiner les divers franchissements possibles rendant vraie la proposition logique associée à un événement. Dans ce sens le grafcet de la figure III.5 n'est pas de type 1 car le déclenchement d'un événement *EV1,* associé à t_1 indépendamment des événements correspondants aux autres transitions, entraîne la réactivation de l'étape 2.

Fig. III.5

Il est à noter qu'un grafcet de type 1 coïncide avec un réseau de Petri « sauf, et sans conflit ». Pour un tel grafcet, les règles de complément n'ont pas lieu d'exister, et il est donc possible de lui appliquer aussi bien des traitements synchrones et asynchrones. Notons d'ailleurs que, dans ce dernier cas, les conflits qui subsisteraient sont éliminés par hiérarchisation.

Remarque : cette définition est volontairement restrictive. Il existe d'autres cas qui autorisent un traitement asynchrone. Une classification précise nécessite une procédure parfois longue à appliquer. Le lecteur intéressé par ce sujet consultera utilement l'ouvrage de M. BLANCHARD sur le grafcet.

Grafcet de type 2 : est un grafcet général nécessitant l'utilisation des règles de complément. Il ne peut donc être mis en œuvre que par traitement synchrone ou parallèle. Etant donné la difficulté d'acquérir une parfaite maîtrise du grafcet de type 2, parfois plus puissant et efficace mais aussi plus problématique, nous suggérons au lecteur de se limiter au grafcet de type 1, ce qui est toujours possible.

Fig. III.6

Considérons par exemple un ensemble de deux chariots C1 et C2, (figure III.6) initialement en A1 et A2 respectivement. Sur un ordre M, les deux chariots se mettent en marche vers la droite. Le premier arrivé en B1 ou B2 attend l'autre pour repartir vers la gauche après obtention de l'autorisation R.

Le redémarrage du processus ne peut se faire qu'avec les deux chariots en A1 et A2, et si le bouton M a été relâché. Les 2 grafcets correspondants sont donnés figure III.7.

Cet exemple nous servira d'illustration au cours de ce chapitre.

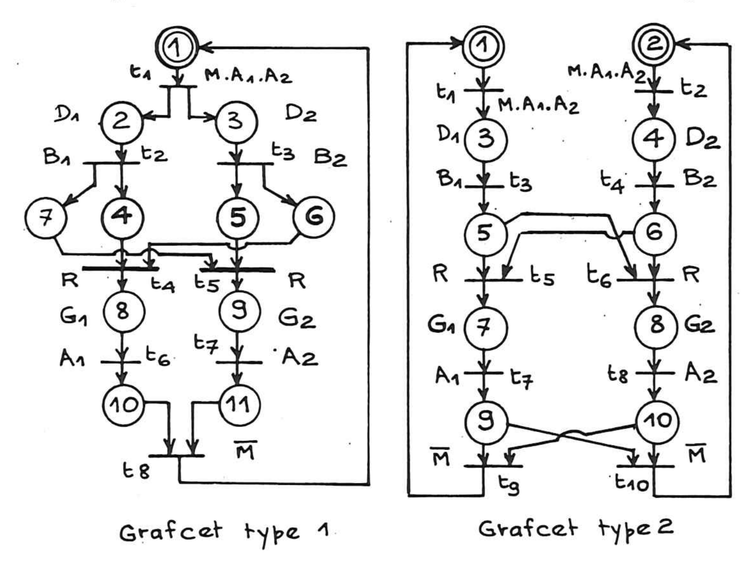


Fig. III.7

III.3. Essai de classification des méthodes

III.3.1. Mémorisation des étapes

L'ensemble des méthodes décrites dans ce paragraphe est basé sur une mémorisation d'une image des étapes d'un grafcet.

Cette mémorisation implique le choix d'un code de représentation permettant d'associer l'ensemble des étapes d'un grafcet à un ensemble de cases mémoires. Cette association peut être quelconque, mais pour réduire l'espace mémoire nécessaire, il est possible d'utiliser un ensemble de n cellules mémoires pour représenter 2^n étapes. Toutefois le coût actuel des mémoires, les contraintes de maintenance et de dépannage du système pro-

grammé conduisent (sauf exception) à l'utilisation d'une cellule mémoire par étape, et c'est ce codage qui sera utilisé dans ce qui suit.

III.3.2. Classification

Nous avons tenté de représenter dans un espace à trois dimensions les diverses méthodes. Un axe concerne la notion de synchronisme ou d'asynchronisme. Nous y avons ajouté la classe des systèmes autosynchrones pour lesquels le traitement de type synchrone ou asynchrone n'est déclenché qu'à l'apparition d'une variation d'entrée ou au passage à un d'un événement.

Le second axe de la figure III.8 est relatif au jeu d'intructions du système programmé utilisé, et à la présence (ou l'absence) d'instructions de branchement.

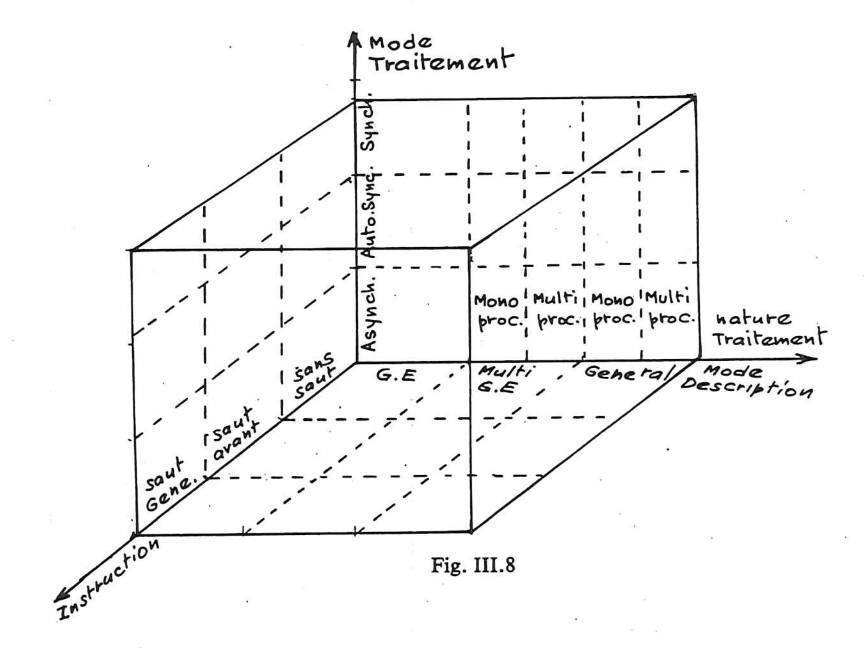


Fig. III.8

On y distingue les machines sans instructions de branchement, celles avec possibilité de branchement vers l'avant (par incrémentation de l'adresse de l'instruction) et enfin les machines universelles permettant des branchements à n'importe quel endroit de la mémoire de programme.

Le dernier axe est affecté d'une part au mode de description, et d'autre part à la nature du traitement.

La première catégorie est relative à une description d'un graphe d'état unique, c'est-à-dire un seul grafcet de type 1, sans simultanéité, ne

comportant à un instant donné au maximum qu'une seule étape active. Cette catégorie permet indifféremment une matérialisation synchrone ou asynchrone.

Une deuxième catégorie correspond à une description par plusieurs graphes d'état indépendants, ou obtenus par décomposition d'un grafcet et un traitement utilisant un ou plusieurs processeurs.

La troisième catégorie est liée à une description par un grafcet général de type 1 ou 2, et à un traitement par un ou plusieurs processeurs.

Notons enfin que les implantations en mémoire, résultats de ces méthodes, ne découlent pas nécessairement directement du travail de l'utilisateur qui peut disposer de moyens importants d'aide à la conception.

III.3.3. Les diverses parties du traitement

Tout traitement d'un grafcet va se décomposer en sept parties que nous indiquons dans un ordre quelconque :

- Acquisition des entrées
- Affectation des sorties
- Initialisation
- Calcul des conditions d'évolution
- Evolution du marquage, ou de l'activité des étapes
- Combinatoire local
- Combinatoire général

Suivant le type de matériel, certaines parties peuvent être traitées ailleurs, soit par un dispositif câblé, soit par un autre processeur. La structure minimale ne garde que l'initialisation et l'évolution de l'activation.

L'initialisation correspond en général à la vérification d'un certain état des entrées et à une autorisation de passage en mode de contrôle automatique. On notera, dans ce qui suit, I la commande correspondante. Lorsqu'elle est présente, elle doit entraîner l'activation des étapes initialement actives et la désactivation des autres. Il convient dès cet instant de traiter le combinatoire local, c'est-à-dire les sorties associées aux étapes, et le combinatoire général indépendant de l'activité du graphe, et destiné à calculer des fonctions logiques particulières, (du type fonction de signalisation par exemple).

A chaque transition t_i est liée une condition d'évolution CE_i produit logique de l'événement associé par l'activité de toutes les étapes d'entrée de la transition.

Enfin il est à remarquer que lorsqu'on travaille avec en mémoire une image des sorties et un déroulement cyclique, une remise à zéro automatique en début de cycle de cette image permet de gagner un grand nombre d'instructions.

III.4. Méthodes pour machines sans instruction de saut

La seule manière de procéder lorsqu'on ne dispose pas d'instruction de saut est de se ramener à un traitement du type combinatoire. Dans le traitement des graphes d'état, les procédures de synthèse sont classiques, c'est pourquoi nous ne présentons ici que deux méthodes, l'une asynchrone, l'autre synchrone qui s'appliquent au grafcet et donc à fortiori au graphe d'état.

III.4.1. Appel - réponse

Il s'agit de la transcription programmée de la méthode classique utilisée en logique câblée dans le cas du choix d'un codage canonique. Elle est donc purement asynchrone et concerne le grafcet de type 1. Le traitement structuré comme sur la figure III.9 procède étape après étape, c'est-à-dire cellule mémoire par cellule mémoire. La condition d'appel d'une étape dénotée a_i, qui correspond à la mise à un de la mémoire associée est donnée par la somme logique des conditions d'évolution de toutes les transitions d'entrée de l'étape.

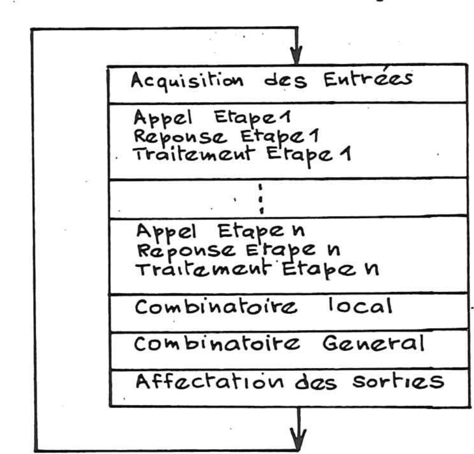


Fig. III.9

La réponse qui correspond à la mise à zéro de la mémoire associée nécessite l'énumération des étapes de sortie des transitions de sortie de l'étape en cours de traitement. Ainsi pour l'étape k de la figure III.10, la réponse r_k est égale à :

$$r_k = y_p + y_q \cdot y_r$$

où y_i représente la valeur prise par la mémoire y_i associée à l'étape i.

Le traitement consiste à réaliser pour chaque étape i une mémoire y_i à priorité à l'arrêt :

$$y_i = (ai + yi) \cdot \overline{ri}$$

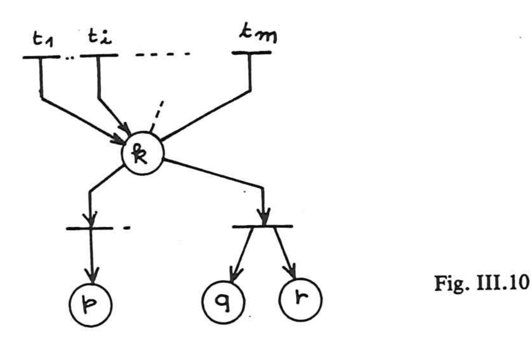


Fig. III.10

On pourrait d'ailleurs aussi bien utiliser une mémoire à priorité à la marche.

L'initialisation se fait par l'union de la commande I à chaque équation d'étape initialement active et par la conjonction par la commande $\overline{I}$ de chaque équation d'étape initialement inactive.

L'évolution complète (activation, désactivation) peut dans certains cas se faire sur deux cycles donnant une possibilité d'aléas. Le traitement se faisant autour des étapes, les actions devront obligatoirement leur être liées, le résultat est un programme important avec une structure de données faible, ne comprenant que les mémoires d'étape.

Cas particulier : dans tous les cas, l'appel et la réponse doivent être disjonctifs. Or lorsque dans un grafcet, on a une boucle ne comportant que deux étapes (fig. III.11.), cette condition n'est pas réalisée avec la procédure précédente. Pour tourner la difficulté, on écrit :

$$a_1 = y_2.EV_2 \quad a_2 = y_1.EV_1 \quad r_1 = y_2.EV_1 \quad r_2 = y_1.EV_2$$

Fig. III.11

Application : Considérons le grafcet de type 1 de la figure III.7.

$a_1 = CE_8\ ;\ a_2 = CE_1\ ;\ a_3 = CE_1\ ;\ a_4 = CE_2\ ;\ a_5 = CE_3\ ;$
$a_6 = CE_3\ ;\ a_7 = CE_2\ ;\ a_8 = CE_4\ ;\ a_9 = CE_5\ ;\ a_{10} = CE_6\ ;\ a_{11} = CE_7$
$r_1 = y_2.y_3\ ;\ r_2 = y_4.y_7\ ;\ r_3 = y_5.y_6\ ;\ r_4 = y_8\ ;\ r_5 = y_9\ ;$
$r_6 = y_8\ ;\ r_7 = y_9\ ;\ r_8 = y_{10}\ ;\ r_9 = y_{11}\ ;\ r_{10} = y_1\ ;\ r_{11} = y_1.$

Le combinatoire local consiste à écrire :

$$D_1 = y_2\ ;\ D_2 = y_3\ ;\ G_1 = y_8\ ;\ G_2 = y_9.$$

III.4.2. Activation - désactivation

Il s'agit d'un traitement synchrone ou asynchrone qui considère pour chacune des étapes d'une part la condition d'activation F_i, somme logique des conditions d'évolution des transitions d'entrée de l'étape *i*, d'autre part la condition de désactivation G_i, somme logique des conditions d'évolution des transitions qui vont sortir de l'étape *i*.

Pour que le traitement soit synchrone, on commence par faire le calcul de tous les F_i et G_i, puis le traitement de chaque étape consiste à réaliser une mémoire « priorité à la marche » pour tenir compte de la règle RC2. On aura ainsi

$y_i = F_i + \overline{G_i}.y_i + I$ pour les étapes initialement actives

et

$y_i = (F_i + \overline{G_i}.Y_i).\overline{I}$ pour les autres.

Le traitement se faisant autour des étapes, les actions doivent forcément leur être associées. Le déroulement purement cyclique se fait sur un seul cycle évitant les aléas de fonctionnement. L'implantation résultante consiste en un programme spécifique, et en une structure de données comportant la mémorisation d'une part de l'activité des étapes, d'autre part des F_i et G_i.

La structuration du programme en mémoire est donnée par la figure III.12.

Acquisition des entrées
Calcul des F_i (i=1...n) Calcul des G_i (i=1...n)
Traitement des Etapes $y_i = F_i + \overline{G_i}.y_i + I$ ou $y_i = (F_i + \overline{G_i}.y_i).\overline{I}$
Combinatoire local
Combinatoire general
Affectation des sorties

Fig. III.12

Application : Soit le grafcet de la figure III-7 (Type II)

$F_1 = CE_9\ ;\ F_2 = CE_{10}\ ;\ F_3 = CE_1\ ;\ F_4 = CE_2\ ;\ F_5 = CE_3\ ;\ F_6 = CE_4;$
$F_7 = CE_5\ ;\ F_8 = CE_6\ ;\ F_9 = CE_7\ ;\ F_{10} = CE_8.$

$G_1 = CE_1\ ;\ G_2 = CE_2\ ;\ G_3 = CE_3\ ;\ G_4 = CE_4\ ;\ G_5 = CE_5 + CE_6\ ;$
$G_6 = CE_5 + CE_6\ ;\ G_7 = CE_7\ ;\ G_8 = CE_8\ ;\ G_9 = CE_9 + CE_{10}\ ;$
$G_{10} = CE_9 + CE_{10}.$

III.5. Méthodes avec instruction de saut vers l'avant

Dans certains cas, en particulier sur machines à déroulement cyclique de la mémoire, on dispose d'une instruction permettant des sauts conditionnels d'instructions mais en autorisant seulement une incrémentation du compteur ordinal.

III.5.1. Le graphe d'état

Le graphe d'état se rencontre lorsque le processus ne nécessite pas de traitement simultané, mais il est à noter que tout grafcet comportant des branches parallèles peut se ramener à un graphe d'état en considérant son graphe d'activité, ou graphe des marquages. Cette transformation n'est toutefois pas intéressante en général car elle accroît la taille du graphe et elle fait perdre le sens physique de la commande du processus à automatiser.

III.5.1.1. Incrémentation conditionnelle

Le contrôle par un graphe d'état peut se faire uniquement à base de l'instruction :

ETAT : SORTIES : EV1 ; EV2 ; INC.

où l'état représente une adresse mémoire à laquelle correspond un certain nombre de commandes. Etant dans cet état si l'événement EV1 survient on passe à l'état suivant par simple incrémentation (+ 1) de l'adresse, si EV2 survient, l'incrémentation est alors de + n, n étant défini par INC. Dans le cas où ni EV1, ni EV2 ne se produisent, on reste dans l'état présent. Le système est donc du type asynchrone.

Le fait pour un état d'avoir au maximum deux successeurs facilite l'écriture mais peut nécessiter une transformation du graphe (fig. III.13). Avec uniquement des incréments positifs, il peut y avoir un problème lors de la numérotation des états. On a parfois à numéroter une étape dont les successeurs ont déjà été numérotés. Dans ces cas critiques, l'étape doit être dédoublée. Ce cas difficile se pose en particulier quand on arrive au dernier état de la mémoire et qu'il faut reboucler sur le début de celle-ci. Le rebouclage peut être automatique par saturation du compteur d'adressage, si un des successeurs du dernier état est le premier, et si le nombre de mots de la mémoire est strictement égal au nombre d'états. S'il n'en est pas ainsi, il convient de dédoubler les successeurs et d'ajouter l'événement certain λ pris comme événement EV2.

Pour illustrer cette implantation, considérons l'exemple correspondant au grafcet de la figure III.14.

L'étape 2, ayant trois successeurs, doit être modifiée. Par ailleurs, le rebouclage de l'étape sur d'autres, précédemment numérotées, entraîne

d'autres transformations. Ceci nous amène à traiter effectivement le grafcet de la figure III.15.

Fig. III.13

Fig. III.14

Fig. III.15

Si nous supposons une mémoire comportant 8 mots (compteur ordinal modulo 8), nous pouvons mettre le programme sous la forme de la figure III.16. Il convient de noter que cette procédure traite des événements globalement. Ceux-ci doivent donc être élaborés par ailleurs. Il est toutefois possible, comme il a été vu au chapitre I, d'intégrer le calcul des événements dans le programme en se servant du jeu d'instructions.

L'initialisation se fait en mettant à zéro le compteur d'état.

Etat	EV(+1)	EV(+n)	n
0	Ev1	–	–
1	Ev2	–	–
2	Ev3	Ev7	+2
3	–	Ev6	+6
4	EV4	Ev5	+4
5	–	λ	+4
6	–	–	–
7	–	–	–

Fig. III.16

III.5.1.2. Pas à pas généralisé

Un graphe d'état n'a au plus qu'une étape active. Lors de l'évolution, l'activité est transmise à une étape lorsque la condition d'évolution associée à une de ses transitions d'entrée est vraie. Comme il n'existe au plus qu'une seule condition d'évolution vraie à un instant donné, le traitement se résume, lorsqu'une condition d'évolution est satisfaite à recopier, pour chaque variable interne Y_i représentative de l'étape *i*, la somme logique des conditions d'évolution associées aux transitions d'entrée.

Dans le cas de l'exemple précédent (fig. III.14) l'organigramme de traitement est celui de la figure III.17.

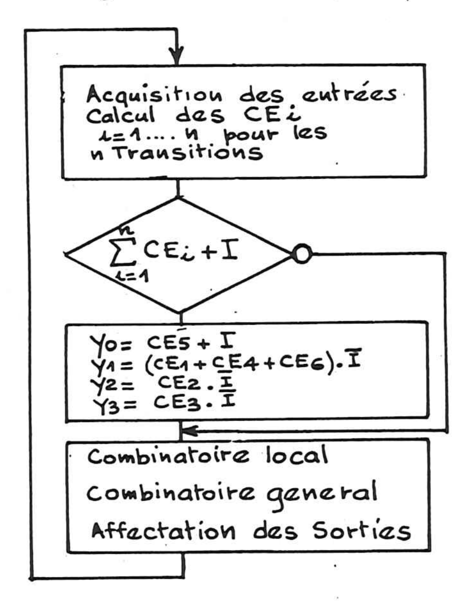


Fig. III.17

La procédure d'initialisation doit fixer Y_0 à 1, et les autres Y_i à zéro, lors de l'apparition de la commande I.

III.5.2. Systèmes multigraphes d'état

Le contrôle d'un processus logique peut faire apparaître soit un ensemble de graphes d'état indépendants, soit, à partir d'une décomposition fonctionnelle, un ensemble avec couplage. Par ailleurs il est possible d'appliquer à un grafcet avec déroulement simultané une procédure de décomposition de façon à obtenir un ensemble de graphes d'état. Remarquons ici qu'il faut se méfier d'une pratique courante qui, dans le cas d'un grafcet de type 1, consiste à affirmer que les composantes connexes obtenues après suppression des transitions de type ET, c'est-à-dire liées à la simultanéité, sont des graphes d'état. L'exemple de la figure III.18 montre le caractère erroné de cette affirmation. Il s'agit d'un réseau de Petri propre, vivant et sauf qui donne, par suppression des nœuds ET, les blocs $\{\overline{P_1}\,;\,\overline{P_6}\,;\,\overline{P_2,P_3,P_4,P_5}\}$. Il est toutefois évident que p_2 et p_3 peuvent être simultanément marquée par t_1. D'autres procédures existent pour effectuer cette décomposition.

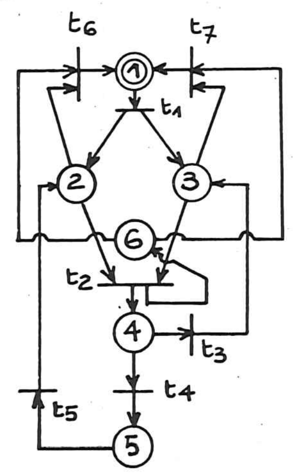


Fig. III.18

Pour gérer un ensemble de graphes d'état à partir d'une structure orientée programme n'utilisant que des sauts vers l'avant, il est possible de mettre en séquence les traitements relatifs à chaque organigramme du type de celui de la figure III.17. Ce mode de travail du type synchrone s'effectue sur un seul cycle. Il permet de considérer des graphes où les actions sont associées aux places ou étapes.

Pour illustrer la procédure, utilisons le cas du grafcet de type II de la figure III.7. Celui-ci se décompose par exemple suivant la partition des étapes $\{\overline{1,3,5,7,9}\,;\,\overline{2,4,6,8,10}\}$ en 2 graphes d'état. A chaque bloc de partition sont associées les transitions d'entrée et de sortie de toutes les étapes du bloc. On en déduit facilement l'organigramme de traitement (fig. III.19) en appliquant les indications du paragraphe 3.5.1.2.

III.5.3. Réseau général

III.5.3.1. Mises à 1 et à 0 conditionnelles

Sur certains automates programmables, il existe des instructions permettant la mise à 1 ou à 0 d'une variable interne ou d'une variable de sortie

en tenant compte d'une condition, fonction logique précédemment élaborée. Celle-ci peut d'ailleurs servir pour plusieurs instructions mises en cascade. L'organigramme équivalent à ce traitement est donné figure III.20. Il montre que l'on dispose en fait d'un saut particulier vers l'avant.

Il devient alors possible de reprendre par exemple la méthode d'appel réponse du paragraphe 3.4.1. pour le traitement asynchrone, et de la reconfigurer en fonction de ces instructions (fig. 3.21). Ceci évite d'avoir à mettre en œuvre les équations des cellules de mémoire. L'ordre dans lequel on traite la mise à 1 ou à zéro permet d'orienter la mémoire en priorité à la marche ou à l'arrêt.

De la même façon on pourrait reconfigurer la méthode synchrone d'activation désactivation du paragraphe 3.4.2.

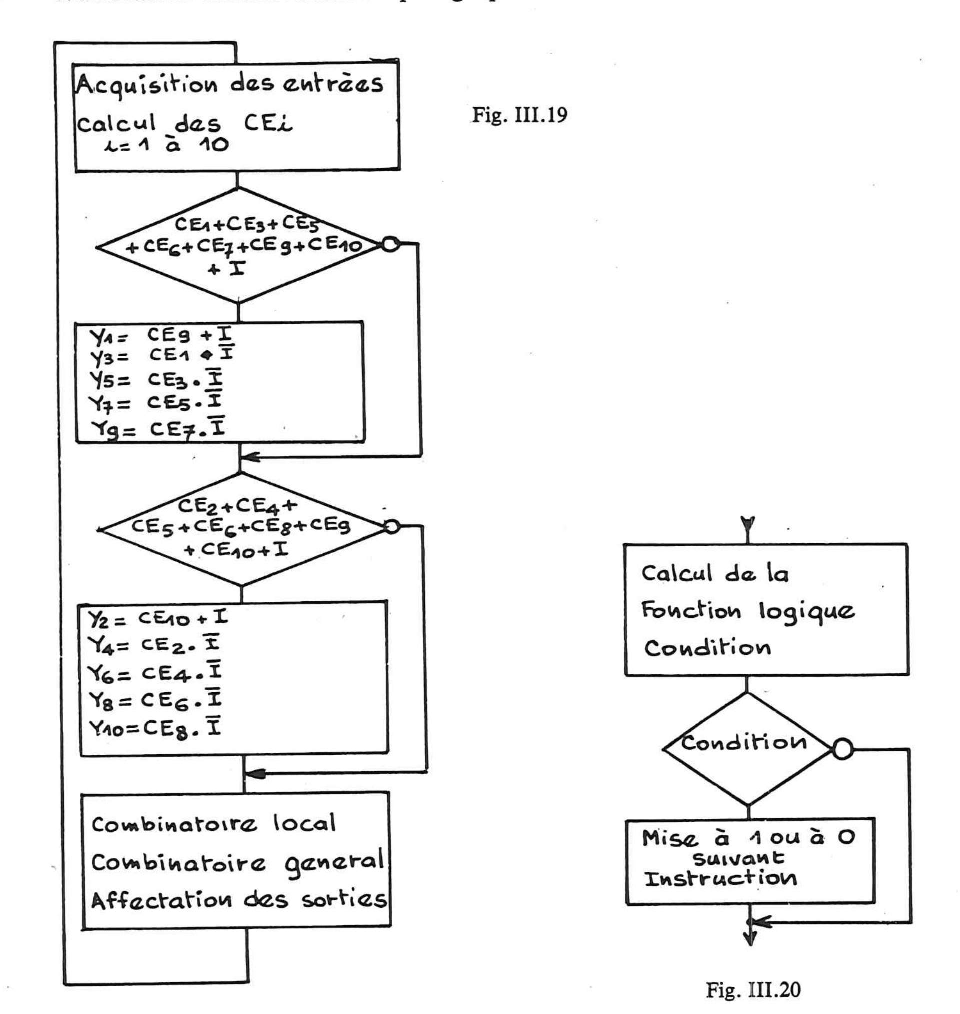


Fig. III.19

Fig. III.20

Acquisition des entrées
Appel Etape 1
Mise à 1 Cond. Etape 1
Reponse Etape 1
Mise à 0 Cond. Etape 1
Appel Etape n
Mise à 1 Cond. Etape n
Reponse Etape n
Mise à 0 Cond. Etape n
Combinatoire local
Combinatoire general
Affectation des sorties

Fig. III.21

III.5.3.2. Traitement sur les transitions

La méthode orientée programme proposée dans ce paragraphe peut s'utiliser aussi bien en mode synchrone qu'en mode asynchrone. Dans le premier cas, on fige tout d'abord les conditions d'évolution puis on traite de l'évolution transition par transition. Dans le cas asynchrone, on calcule pour chaque transition, la valeur de la condition d'évolution puis on fait éventuellement évoluer. Le déroulement se fait évidemment sur un cycle. Le traitement en mode asynchrone est représenté pour le grafcet de type I de la figure III.7 par la figure III.22. Il est obtenu en considérant transition par transition. Les réseaux de type *t* peuvent facilement être traités, de même que ceux de type *s*, en effectuant l'étape de traitement du combinatoire local.

L'initialisation se fait par une transition, source fictive qui force à 1 les étapes initialement actives, et à 0 les autres. Le traitement de l'initialisation doit être prioritaire.

Pour le traitement des grafcets de type II, il est nécessaire de se placer en mode synchrone et de séparer la mise en œuvre de l'évolution en deux parties, de telle sorte que l'activation soit prioritaire, en prenant en compte d'abord les désactivations, puis les activations.

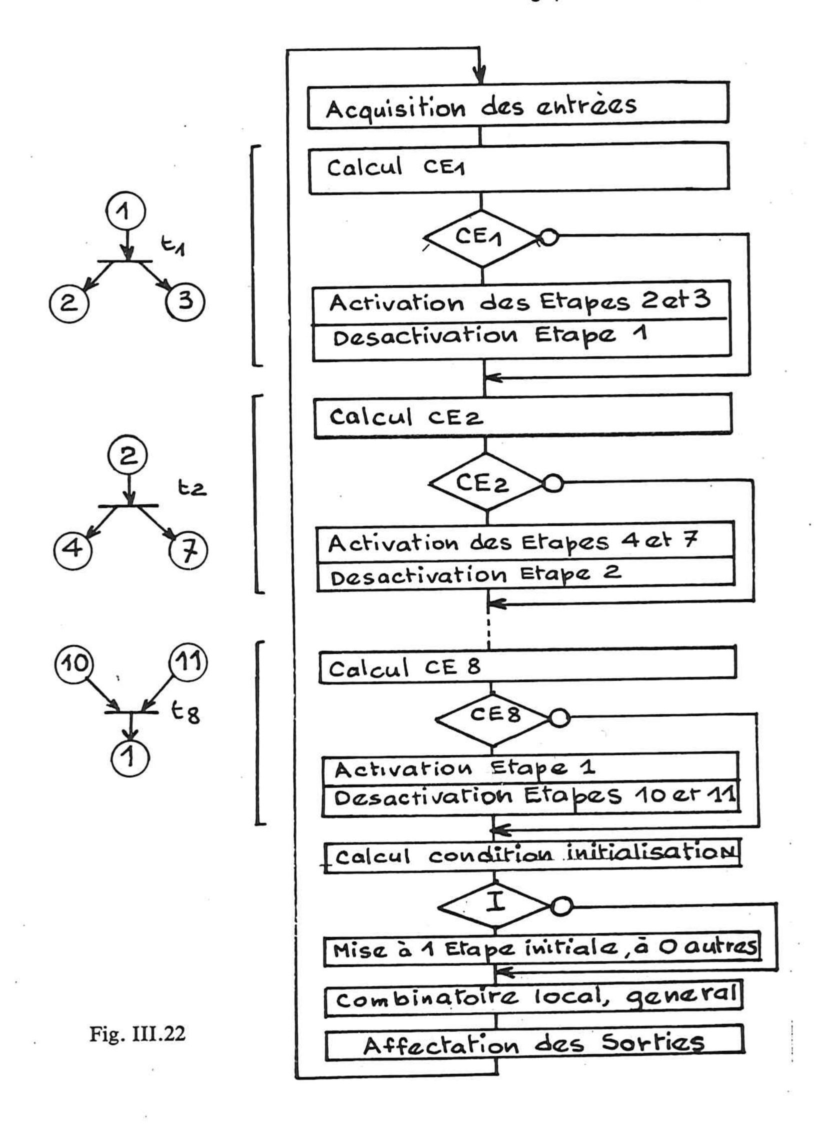


Fig. III.22

III.6 Méthodes avec instruction de saut général

III.6.1. Le graphe d'état

L'instruction de saut général supprime les problèmes de numérotation d'étapes précédemment évoqués car il est possible d'adresser n'importe quel endroit de la mémoire.

La forme, le nombre et la constitution des instructions dépend de la quantité de traitement intégré à l'instruction, mais également du compromis accepté entre la longueur du mot instruction et le nombre d'instructions, donc de la vitesse d'exécution.

L'adresse d'une instruction correspond soit directement au numéro de l'étape soit à un codage de sa référence.

Dans beaucoup de cas, les tests effectués portent uniquement sur une variable et non pas sur la réceptivité sauf si celle-ci est calculée par ailleurs. Toutefois on sait par le premier chapitre qu'il est facile de traiter le combinatoire en multipliant les étapes fictives, c'est-à-dire en le transformant en séquentiel. Considérons par exemple un jeu d'instructions comportant trois instructions :

TYPE S : SORTIE - VALEUR
TYPE T : TEST VARIABLE - ADRESSE VRAIE - SORTIE IMPULSIONNELLE
TYPE I : SAUT A ADRESSE - SORTIE IMPULSIONNELLE

La première permet de traiter les graphes de type S. On y fait figurer la référence de la sortie ou de la variable interne affectée ainsi que la valeur qui lui est attribuée. L'instruction de test assure, suivant la valeur de la variable testée, le branchement à l'adresse indiquée, ou le passage à l'instruction suivante. Il est possible d'associer à ces instructions une sortie dans le cas d'une description par un graphe de type t.

Le dernière instruction est un branchement inconditionnel. Pour traduire dans cette écriture, le morceau de graphe d'état représenté figure III.23.a, on considère en fait la transcription sous la forme III.23.b et on en déduit le programme de la figure III.24.

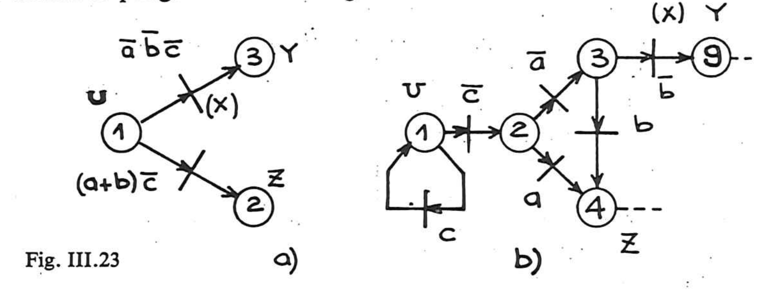


Fig. III.23

Adresse	Type	
0	S	U
1	T	c - 1 -
2	T	a - 4
3	T	b̄ - 9 . X
4	S	Z
⋮	⋮	⋮
9	S	Y

Fig. III.24

D'autres jeux d'instructions sont possibles :
On peut envisager pour mettre en œuvre un graphe de type t la seule instruction :
TEST VARIABLE - ADRESSE SI VRAI - ADRESSE SI FAUX - SORTIES SI VRAI - SORTIES SI FAUX.

Mais on peut aussi décomposer cette instruction en un ensemble de trois instructions successives :

TEST D'UNE VARIABLE
SORTIES ET ADRESSE SI VRAI
SORTIES ET ADRESSE SI FAUX

Il est également possible de créer des instructions avec tests comprenant des multiples, comme par exemple :

TEST	TEST	ADRESSE	ADRESSE	ADRESSE	ADRESSE	SORTIES
X	Y	$\overline{X}\,\overline{Y}$	$\overline{X}Y$	$X\overline{Y}$	XY	PAR CHAMP DE TEST

Pour ne pas avoir de mots trop longs, tout en ayant un champ d'adresse suffisant, on peut faire de l'adressage relatif. Pour cela on se réfère à un registre interne qui constitue le point fort de l'adresse. Le registre est généralement modifiable par un ensemble particulier d'instructions.

Une autre idée intéressante pour traiter les graphes de type t est de considérer, à chaque pas de traitement, la liste des transitions validées. On réalise donc une structure «orientée données» telle qu'à chaque adresse de transition, on dispose de la fonction combinatoire de réceptivité à tester, de la nouvelle liste de transitions à examiner en cas de franchissement, ainsi que des actions qui lui sont associées. Un programme de gestion doit être réalisé de façon à utiliser les données de cette table constante représentative du graphe et de la table dynamique des transitions validées.

Pour illustrer cette idée, considérons de nouveau le graphe d'état de la figure III.14.

Après une phase d'initialisation nécessaire pour définir la liste initiale des transitions à traiter, le programme de gestion entre dans une boucle qui, après prise en compte de la valeur des entrées, examine une à une les transitions validées, données par leur adresse de position dans la table du graphe.

On y rencontre tout d'abord l'adresse du sous-programme de traitement de l'événement associé. Le résultat à 0 de cet événement conduit à examiner la transition suivante si celle-ci existe, sinon on retourne au début du programme pour un nouvel examen de l'ensemble des transitions. Lorsque l'événement est validé, la transition correspondante peut être franchie, et les sorties associées sont appliquées. Le programme moniteur de la figure III.25 doit recopier jusqu'à l'indication FL de fin de liste, la nouvelle liste de transitions.

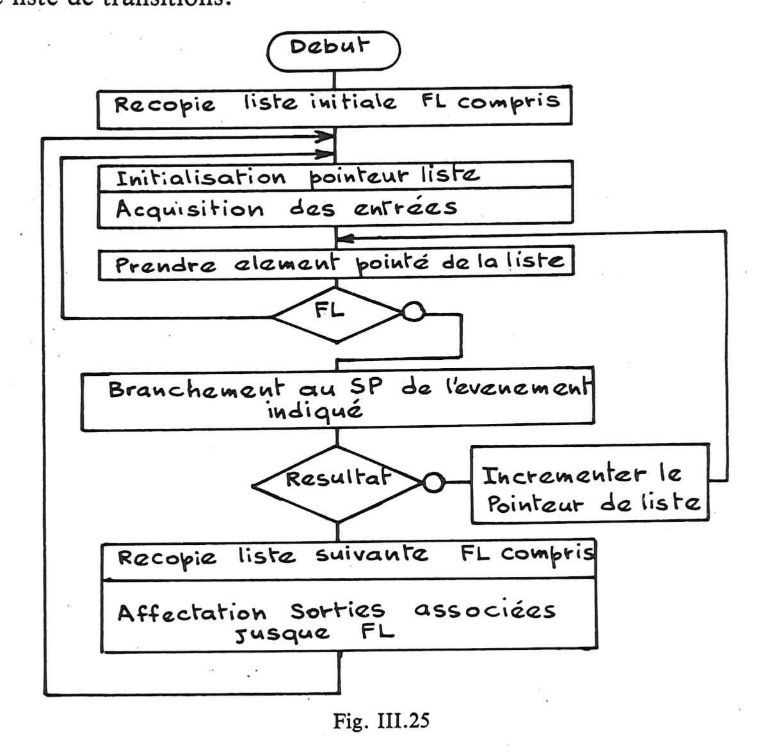


Fig. III.25

La table des données relative au niveau s'écrit sous la forme générale de la figure III.26 ce qui pour notre exemple donne celle de la figure III.27 dans laquelle les listes de sorties se réduisent à l'indication de fin de liste FL.

La méthode précédente, qui nous a permis d'élaborer une structure « orientée donnée », avait pris comme point d'examen la transition. Il est possible de faire la même chose avec la place. Comme il ne peut y avoir qu'une seule place active, il suffit d'une seule case adresse de traitement, le pointeur courant. Le programme moniteur de la figure III.28 gère la structure de données dont la forme est définie par la figure III.29. Appliqué à l'exemple précédent, on obtient la table de la figure III.30. La structure obtenue permet de traiter plus aisément les graphes du type S, mais il est possible d'adjoindre, lors du franchissement des transitions, des ordres pour traiter des éléments de type t. Le traitement du combinatoire local se fait dans l'organigramme de la figure III.28 chaque fois qu'il n'y a aucune évolution dans le graphe. Ceci est intéressant car il est ainsi possible de traiter des commandes conditionnées par les entrées. Cette méthode présente un inconvénient lorsqu'une évolution doit être effectuée.

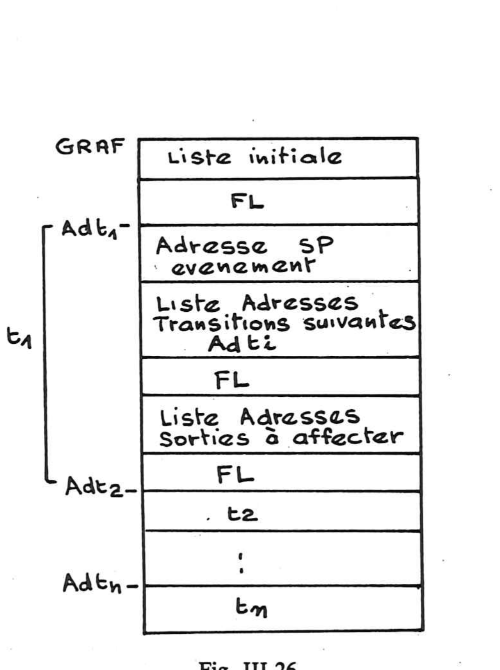


Fig. III.26

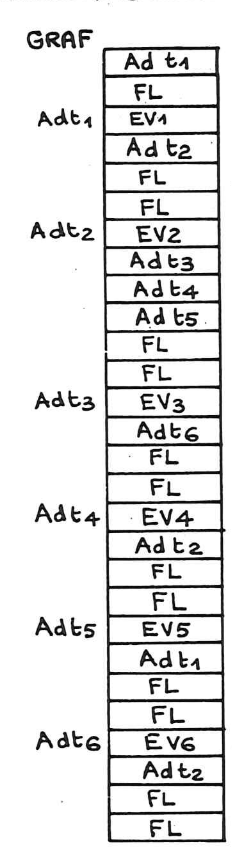


Fig. III.27

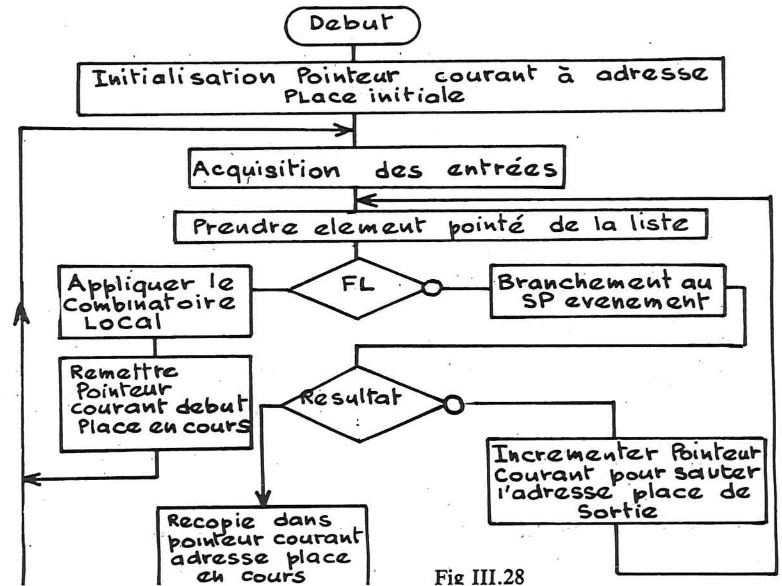


Fig III.28

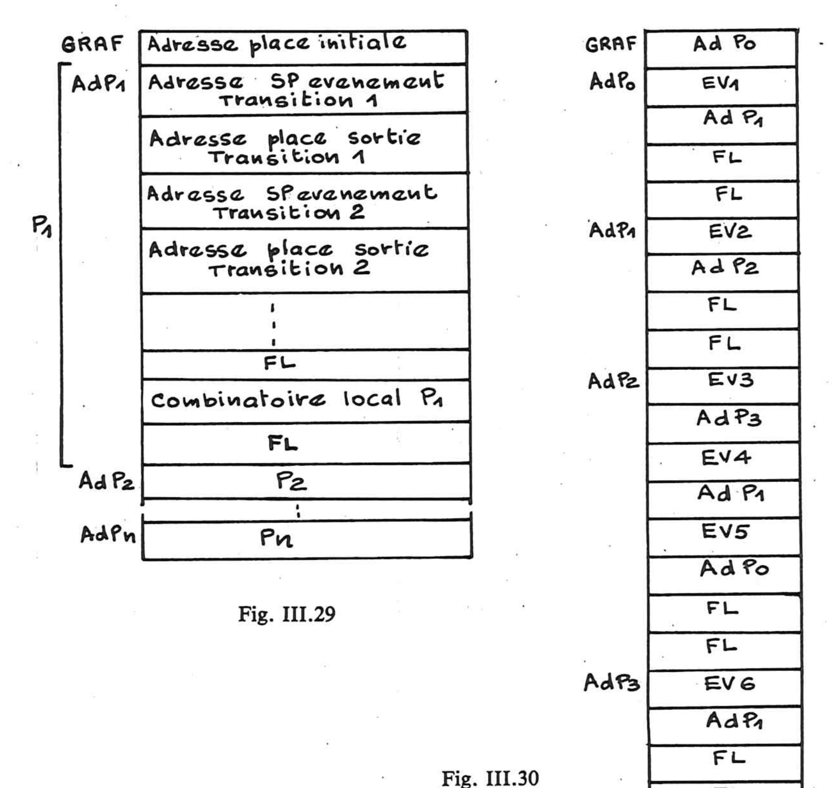


Fig. III.29

Fig. III.30

En effet la situation dans l'organigramme du traitement du combinatoire local interdit la description de graphes avec des places transitoires comportant des opérations combinatoires, car celles-ci ne seraient pas prises en compte.

On pourrait modifier l'organigramme soit en ajoutant un traitement de combinatoire local aussitôt après recopiage dans le pointeur courant de l'adresse de la place de sortie, soit en déplaçant le traitement du combinatoire local de la figure III.28 juste avant la prise de l'élément pointé dans la liste. Cela impose, dans la structure de données, de placer pour chaque étape le combinatoire local en début de liste, et dans le programme, de repositionner le pointeur courant, avant d'appliquer le combinatoire.

Les organigrammes précédents comportent une boucle de scrutation qui peut être parcourue un grand nombre de fois sans qu'il y ait un changement dans les entrées. Il en résulte un gaspillage du temps de traitement, qui pourrait être consacré à d'autres tâches. Nous ferons à ce sujet les deux remarques suivantes :

Compte tenu du temps de réponse des automatismes industriels, il est sans importance de travailler très vite, et il est possible de fixer une durée

de cycle utilisant une temporisation, et d'effectuer le traitement principal après le lancement de cette temporisation, puis de passer aux autres tâches, avec retour en début de cycle à la fin de la temporisation.

Une autre idée consiste à profiter des propriétés des structures autosynchrones pour augmenter le temps consacré aux autres tâches. L'examen des changements des entrées permet en effet d'effectuer soit le traitement principal, soit celui des tâches annexes.

III.6.2. Les ensembles de graphe d'état

Une première manière de traiter les ensembles de graphes d'état est évidemment d'associer un des processeurs précédents à chaque graphe, puis d'examiner les couplages entre les divers processeurs. La seule difficulté réelle est alors de garantir un véritable traitement synchrone au niveau de l'ensemble, lorsque cela est nécessaire. Notre propos dans ce paragraphe sera surtout consacré au traitement sur un même processeur d'un ensemble de graphes d'état. Il s'agit en effet de partager le cycle de traitement en sous-cycles destinés au traitement de chaque graphe (parfois dénommé séquence ou branche). Le traitement sera synchrone si les conditions d'évolution sont évaluées et figées avant de parcourir chaque branche. Si les calculs concernant les événements associés aux transitions sont placés dans le traitement d'une branche ou d'une séquence, la machine fonctionne en mode asynchrone et elle est donc limitée au grafcet de type I.

Lorsqu'on effectue la décomposition en graphes d'état, ou même en graphes particuliers correspondant soit à des séquences pures, (successions d'étapes et de transitions), soit à des branches de grafcet ne comportant pas de nœud ET mais seulement certaines structures du type OU, les graphes obtenus ont au plus une étape active à un instant donné, et peuvent même être totalement inactifs donc vides de marqueurs. Il reste toutefois possible d'ajouter à chacun des ces graphes une étape d'attente initialement marquée qui permet la description de la désactivation du graphe.

Ainsi le grafcet de type I de la figure III.7 peut être décomposé en quatre graphes d'état, comme le montre la figure III.31; m_i indique l'activité de l'étape *i*. Les étapes 12, 13 et 14 ont été ajoutées pour que chaque graphe d'état ait une et une seule étape active.

Les structures « orientées données » décrites dans le paragraphe précédent peuvent être facilement généralisées. Nous n'en ferons la présentation qu'en mode asynchrone à partir de la structure de traitement autour des places, l'exemple de la figure III.31 nous servant d'illustration. Chaque graphe d'état sera repéré dans le programme moniteur représenté figure III.32 par son pointeur courant. Il existe donc une table des pointeurs courants qui doivent être initialisés en début de traitement. Chaque graphe d'état est ensuite traité comme au paragraphe précédent, le cycle reprenant lors de la rencontre de l'indicateur de fin de table FPC. La structure de données commence par les adresses des places initiales suivies de l'indicateur FPC. Les éléments concernant les étapes peuvent être placés de n'importe quelle façon, indépendamment de l'appartenance à une branche. Avec un tel traitement, il n'existe nulle part en mémoire de table des marquages, il convient donc de considérer les m_i comme des variables internes, prises en compte par le combinatoire local des étapes *i*. La structure de données de l'exemple de la figure III.31 est donnée figure III.33 pour le premier sous-grafcet.

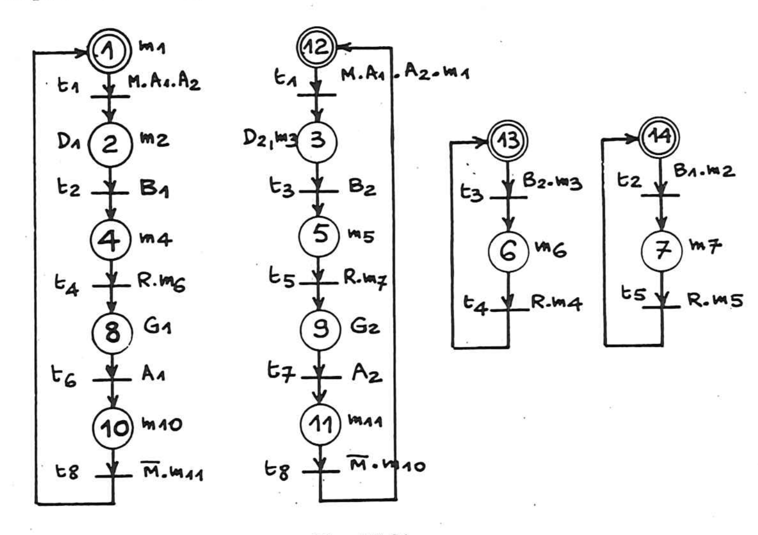


Fig. III.31

Suivant la remarque précédente, nous avons placé le combinatoire local dans la branche principale. Il correspond, d'après la table des données, à une machine de Moore, mais il est possible d'appliquer les principes exposés dans le premier chapitre à propos des machines de Mealy.

Une structure « orientée programme » basée sur la notion « d'indirection » est également envisageable. On utilise encore une table des pointeurs, mais au lieu de résulter d'un programme moniteur, le contrôle est réparti dans le programme de traitement lui-même. La place marquée de chaque graphe d'état est alors repérée par son adresse dans la table des pointeurs. Pour chaque traitement de place il existe plusieurs possiblités de franchissement de transitions, correspondant soit à une incrémentation, soit à une modification du pointeur associé. Il convient de diriger ensuite le traitement, par indirection, à la place marquée du graphe d'état suivant.

La figure III.34 donne l'organigramme de traitement d'une place.

On réalise ainsi un système en temps partagé, à contrôle autoréparti, dont l'unité de base est la place et ses transitions de sortie.

Compte tenu de la variété des commandes et des traitements qu'il est possible d'associer à une place, (calculs, sous-programmes...), cette notion est tout à fait générale et peut s'appliquer en particulier à des traitements numériques multitâches, ou à la supervision en temps réel.

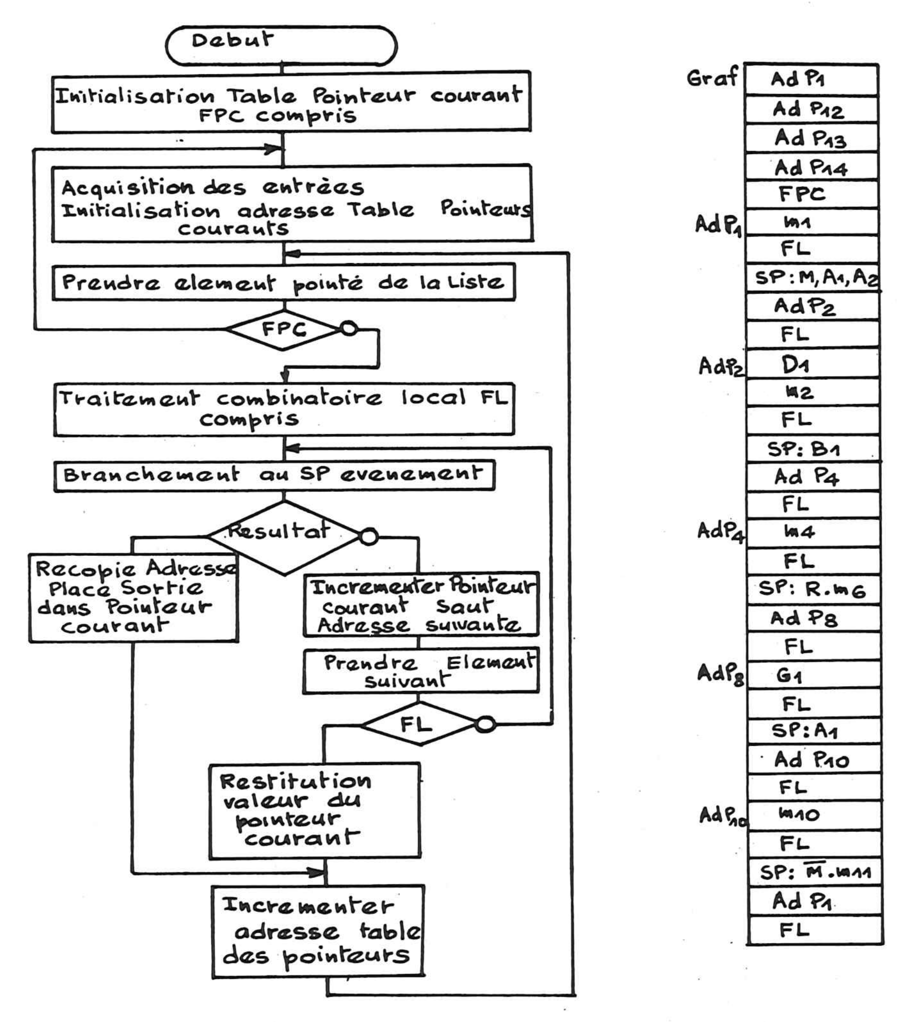


Fig. III.32

Fig. III.33

Par exemple le traitement du cas de la figure III.31 nécessite quatre pointeurs PP_1.... PP_4 : un pour chaque graphe d'état. Ceux-ci sont initialisés avec les adresses des places initialement marquées. Le programme s'écrit en mettant dans n'importe quel ordre le traitement de chaque place.

La figure III.35 montre, pour quelques places, l'organisation du programme. Il est à noter que l'acquisition des entrées, le calcul d'un combinatoire général, ainsi que l'affectation globale des valeurs de sortie, se fait lors du passage du dernier au premier graphe d'état.

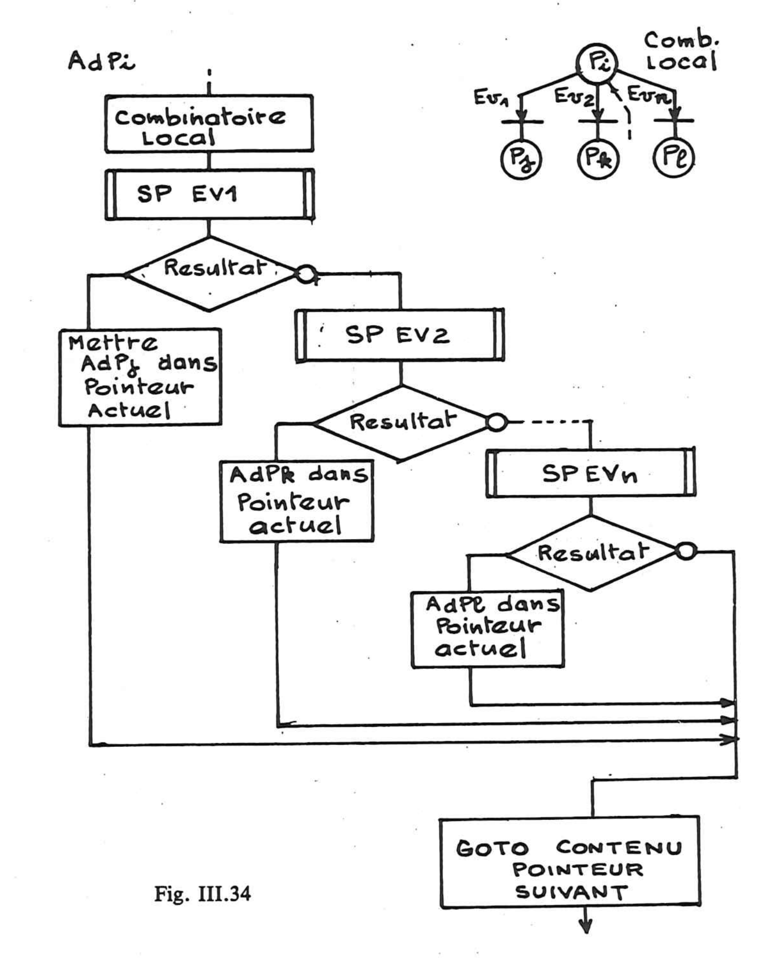


Fig. III.34

III.6.3. Grafcet général

Nous arrivons au traitement du cas le plus général ; le problème se pose de la façon suivante : plusieurs places (ou étapes) appartenant à une même structure, le réseau de Petri ou le grafcet, peuvent être marquées simultanément et induire un certain ensemble de traitement.

Un traitement sur les transitions peut être effectué. Il s'adapte plus particulièrement au réseau du type *t* car il n'est pas nécessaire de garder trace des places marquées. Il convient de mémoriser les transitions validées

à un instant donné. Si au contraire, on considère un traitement sur les places, une table du marquage du réseau est nécessaire.

Quelle que soit la forme du traitement, deux types de tables peuvent être envisagés, soit en associant un indicateur à chaque place ou transition, ce qui entraîne un balayage pour la détection d'un traitement et une gestion plus aisée, soit en constituant une table où sont indiquées uniquement les références des places actives. Cette dernière procédure plus rapide, demande un espace mémoire réduit, mais sa gestion est plus délicate, car la longueur de la table est dynamiquement variable.

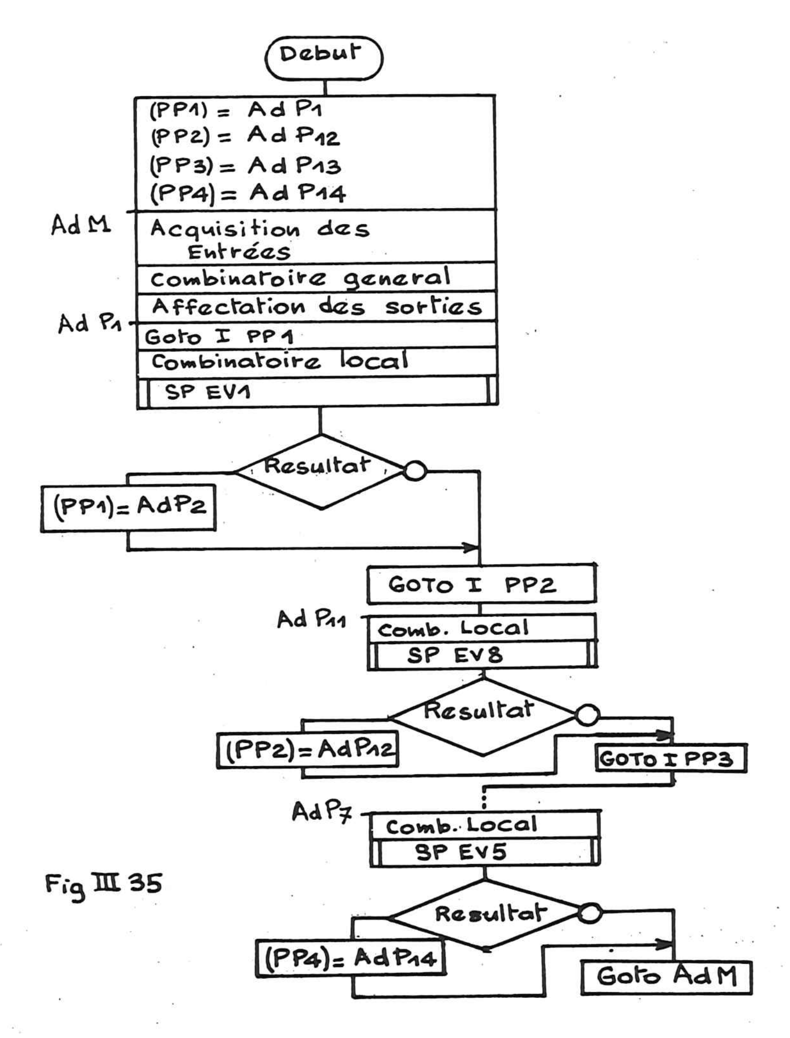


Fig. III.35

III.6.3.1. Traitement sur les places

Structure des données. Considérons ici une structure orientée « données » basée sur la place. La description en mémoire du réseau ou du grafcet reste celle déjà rencontrée lors des autres traitements et peut correspondre par exemple à celle de la figure III.29 à condition d'ajouter pour les transitions du type « jonction », (c'est-à-dire avec plusieurs places d'entrée), les références des places qui interviennent dans la condition d'évolution. Ceci entraîne pour ces transitions une redondance de description et de traitement. On peut en effet soit tester des transitions non validées, soit représenter et tester plusieurs fois la même transition.

Pour remédier à cette difficulté, il est possible d'introduire la notion de « place clé », liée à l'idée qu'une transition doit être représentée une fois et une seule.

Une place clé est une place qui conditionne le franchissement d'un certain ensemble de transitions. Les places d'entrée, autres que la place clé, pour une transition du type jonction sont dites « places de synchronisme ». Leur marquage doit apparaître dans la condition d'évolution de la transition. On représente donc ces places par une variable interne *mj* mise à un quand la place *Pj* est marquée, et à zéro quand elle est démarquée (fig. III.36).

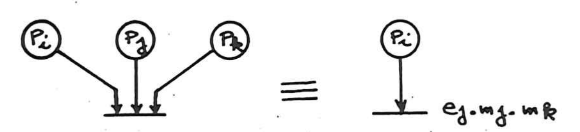


Fig. III.36

La structure de données se limite donc à la description des places clés et de certaines places pour lesquelles on n'indique que le combinatoire local. Le traitement des actions des « places non clés » est effectivement un problème. Si les commandes sont inconditionnelles, il suffit de se ramener à un réseau de type *t* en plaçant des mises à un, ou à zéro, d'actions sur les transitions d'entrée ou de sortie de la place. Pour les actions conditionnelles, il faut : soit augmenter le nombre des places clés comme indiqué plus haut, soit faire un report en combinatoire général, après avoir noté le marquage en variable interne.

La structure de données pour une place *Pi* correspond à la figure III.37.

Le choix des places clés sera fait pour minimiser le temps de traitement, ce qui revient à minimiser leur nombre. Pour cela on commence par détecter les places «obligatoirement clés», car place d'entrée unique d'une transition. Ce premier pas peut entraîner la définition de places de synchronisme pour d'autres transitions. Lorsque plusieurs places clés ainsi définies précèdent une même transition, le choix parmi celles-ci est libre.

Ceci nous montre qu'une place clé pour une transition peut être «de synchronisme» pour une autre transition.
La deuxième étape du choix considère l'ensemble des transitions ne disposant pas encore de place clé. On cherche alors la place antécédente au plus grand nombre de transitions. Elle est alors dite « place clé ». En cas d'égalité le choix est indifférent. Cette deuxième étape est ensuite répétée autant de fois que nécessaire.

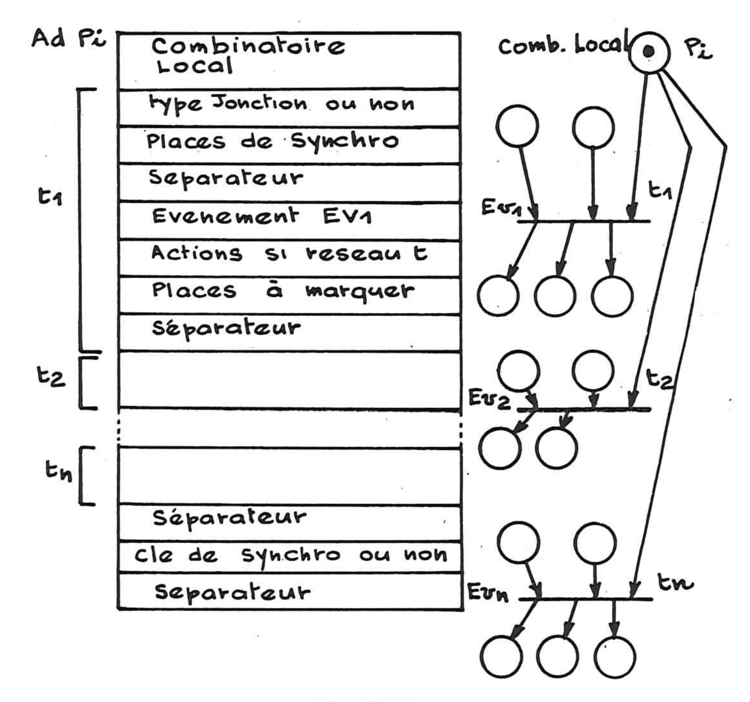


Fig. III.37

A toute place de synchronisme, on associe une variable interne.
Pour illustrer cette notion, considérons l'exemple de la figure III.7. Les étapes 1, 2, 3, 4, 7, 8 sont obligatoirement « étapes clé ». Il reste alors à considérer les transitions $t_5\,t_6\,t_9$ et t_{10}. On pourra prendre les étapes 5 et 9 comme « étapes clés », ce qui donne 6 et 10 « étapes de synchronisme ».

Tables de marquage.

Le traitement sur les places nécessite une représentation du marquage. Dans le cas où on associe un indicateur par place ou étape, il est nécessaire de faire une scrutation, toujours lente, dont la durée est fonction de la taille du réseau. En pratique cette méthode est inexploitable pour les gros réseaux, sauf dans le cas d'une utilisation avec multiprocesseur.

En version monoprocesseur une procédure intéressante consiste à considérer deux tables dynamiques représentatives du marquage, l'une utilisable pour le marquage dans le cycle en cours de traitement et l'autre pour le cycle suivant. A chaque cycle, le rôle des tables est interverti. Le temps d'exploitation est fonction du degré de parallélisme du graphe.

Pour éviter de passer par une table intermédiaire des adresses, la place ou l'étape est repérée directement par son adresse plutôt que par son numéro.

Gestion des tables.

La gestion des tables dépend du type de grafcet à traiter. Dans le cas d'un grafcet de type I, la procédure pourra être asynchrone car il n'y a pas à tenir compte des éventuels conflits. Pour le grafcet général, la gestion doit être telle que la simultanéité de franchissement soit possible, de même que la priorité de l'activation sur la désactivation.

Toute la difficulté provient du fait que certaines places clés sont aussi places de synchronisme. Cette caractéristique essentielle doit figurer dans la table des données.

La procédure asynchrone est la suivante :

Pour toutes les places autres que les places clés de synchronisme, si aucun franchissement de transition n'est possible, on recopie l'adresse de la place dans la nouvelle table. Si on a un franchissement, on copie les places clés de sortie dans la nouvelle table, puis on fait la mise à 1 (0) des variables internes des places de synchronisme de sortie (d'entrée) de la transition, et enfin on passe au traitement de la place suivante, sans considérer les autres transitions de la place en cours de traitement.

Pour illustrer le cas des « places clés de synchronisme », considérons l'exemple de la figure III.38,qui peut être transformé en celui de la figure III.39, où P_2 est à la fois étape clé et étape de synchronisme.

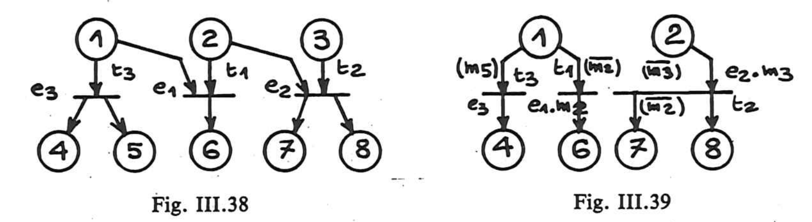


Fig. III.38 Fig. III.39

Une telle place P_i étant représentée par une variable interne m_i, le traitement commence par le texte de cette variable. Si elle est nulle, la place P_i n'est ni traitée, ni recopiée (car une autre transition faisant intervenir P_i en synchronisme a déjà été franchie). Sur l'exemple donné, ce serait le cas pour le franchissement préalable de t_1. Si m_i est à un, le traitement s'effectue. Dans le cas d'un franchissement, on met à jour la nouvelle table et les

m_i par rapport aux places de sortie. Par contre, si aucun franchissement ne se produit, on ne peut recopier P_i car il convient d'attendre tous les traitements de transitions où m_i intervient pour savoir si P_2 doit être démarquée.

A la fin de tous les traitements des places de la table actuelle, on teste les m_i des places clés de synchronisme. Si ceux-ci sont à un, on recopie P_i dans la liste pour le cycle suivant. Un tel traitement s'effectue sur un cycle et est asynchrone car une modification de m_i peut intervenir au cours du cycle.

Si on accepte de travailler sur deux cycles, on peut éviter le test final des places clés de synchronisme. A cette fin, lorsqu'il n'y a pas de franchissement de transitions on recopie P_i dans la nouvelle table. Si ultérieurement un franchissement met m_i à zéro, (ce qui dans l'exemple se produit pour m_2 dans le traitement successif de t_2 non franchie et t_1 franchie)ce fait est pris en compte au cycle suivant par l'impossibilité de traitement de P_i.

Le recopiage d'une place clé de synchronisme entraînant la mise à 1 de la variable m_i associée, il en résulte que si un marquage de P_2 intervient avant son traitement celui-ci s'effectue dans le même cycle au lieu du cycle suivant.

Examinons maintenant le cas du grafcet de type II. La nature synchrone du traitement oblige de figer pour le cycle en cours les valeurs des m_i et la liste des P_i. La détermination de la nouvelle valeur de m_i pour le cycle en cours dépend de la valeur précédente et de deux termes ri et si, (respectivement remise à zéro et à un), à établir au cours du cycle en fonction des ordres (m_i) ou ($\overline{m_i}$) liés aux transitions.

Les *ri* et *si* sont mis à zéro en début de cycle. Le nouvel m_i se calcule à la fin du cycle de façon à donner la priorité à l'activation, donc :

$$m_i = m_i.\overline{r}_i + s_i$$

La liste des P_j est doublée comme dans le cas asynchrone. Le seul fait de l'impossibilité d'effacer une valeur introduite dans cette nouvelle liste donne la priorité à l'activation.

Le moniteur de gestion est représenté par la figure III.40. Il comprend une partie d'initialisation, la prise en compte des entrées et sorties, la gestion des tables. Pour faciliter l'initialisation, il est toujours possible de se ramener à une étape initiale unique. Il est aussi à noter que le combinatoire général peut être représenté dans la table comme une étape supplémentaire toujours active.

Pour plus de clarté notons *PP* le pointeur de la table des étapes actives pour le cycle en cours et *PP** celui de la table pour le cycle suivant. Le contenu d'un élément de la table pointé par *PP* sera noté *(PP)*. En début de cycle de traitement les contenus de la table correspondent aux adresses de début des étapes dans la table de données (notation *(PP) o)*. Les contenus évoluent ensuite pour traiter ce qui concerne les étapes.

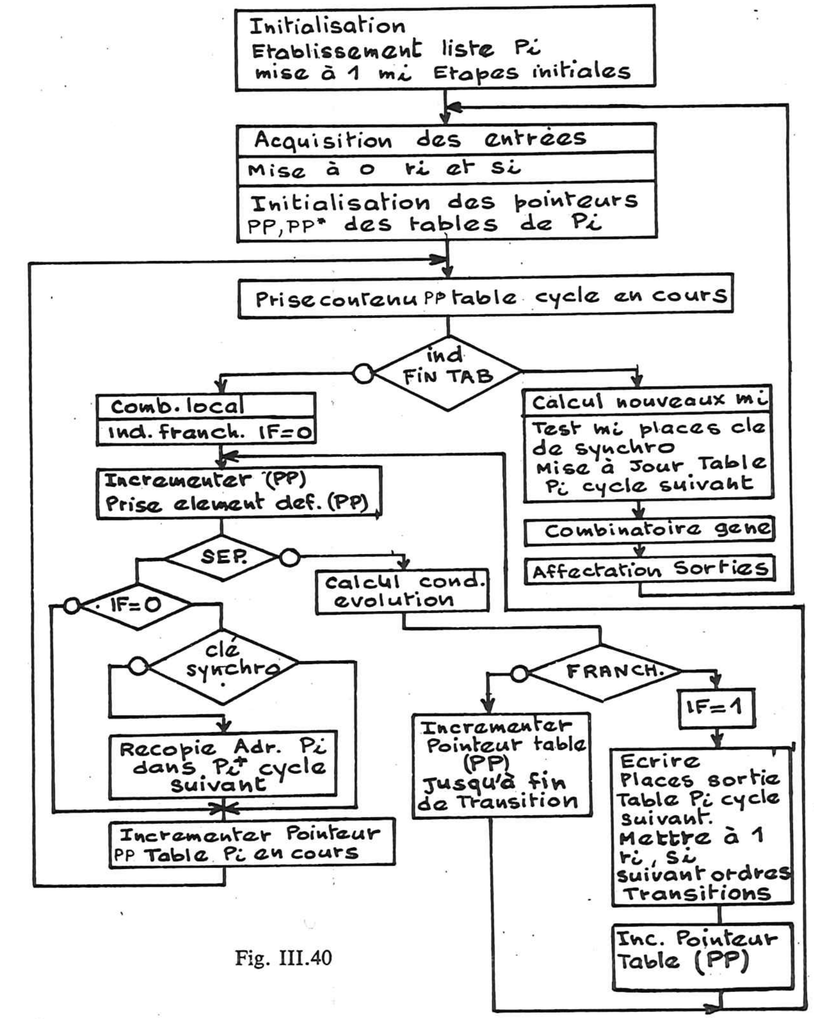


Fig. III.40

Pour les étapes autres que les étapes clés de synchronisme, l'absence de franchissement de transition entraîne le recopiage dans la table des P_i. Lors d'un franchissement de transition la copie des étapes de sortie s'effectue dans P_i, la mise à un des termes s_i et r_i est faite pour les étapes de synchronisme *m,* respectivement de sortie et d'entrée. Par ailleurs cela ne bloque pas le traitement des autres transitions liées à la même étape. Pour une étape clé de synchronisme, le traitement est identique à celui des autres étapes clés sauf au niveau du recopiage qui ne doit s'effectuer qu'après obtention, à la fin de tout le traitement, d'une valeur 1 pour le nouvel m_i correspondant à cette étape clé de synchronisme. Le recopiage est systématique si la place ne figure pas déjà dans la liste.

La structure de données comprend donc, en plus de la description du réseau une table des adresses de places en cours de traitement, une table des adresses de places pour le cycle suivant, des mémoires de un bit pour les termes m_i, r_i, et s_i.

III.6.3.2. Traitement sur les transitions

Il est possible de reprendre dans un traitement autour des transitions tous les éléments relatifs au traitement sur les places. La mise en œuvre procède du même esprit, les réseaux correspondant seront de type *t*.

A un instant donné, un certain nombre de transitions sont validées et figurent dans une liste. Le but du traitement est, après avoir établi les ordres liés aux divers franchissements, de définir la liste des transitions validées pour le cycle suivant. La structure de données est présentée sur la figure III.41.

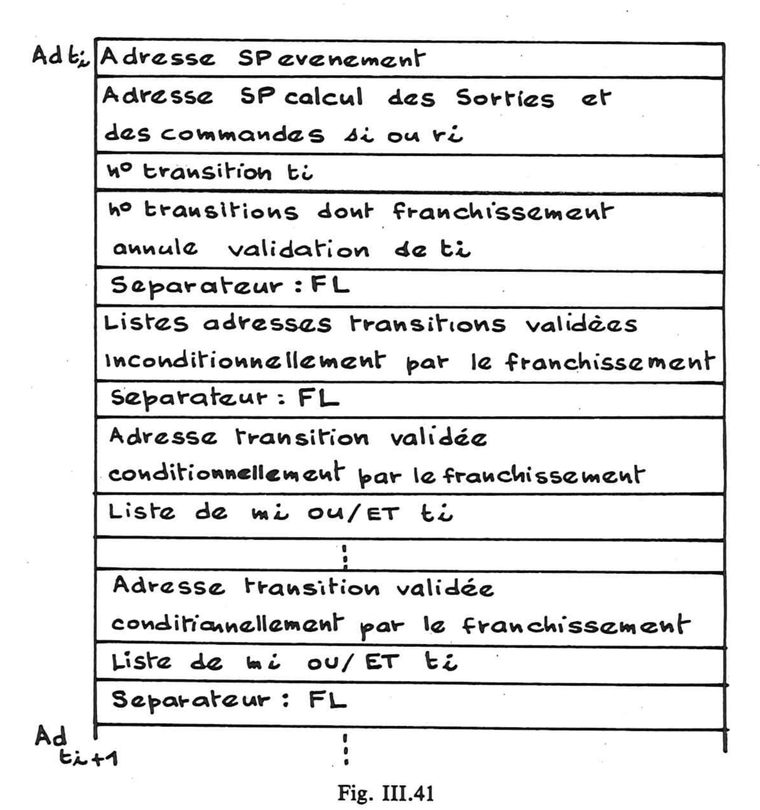


Fig. III.41

A celle-ci viennent s'adjoindre les deux listes de transitions validées pour le cycle en cours et pour le suivant. Les transitions y sont repérées par leur adresse. Par ailleurs, il est nécessaire de disposer d'une liste des transi-

tions franchies, repérées par leur numéro. Enfin nous ajoutons pour chaque transition, qui ont parmi leurs successeurs immédiats une transition du type jonction, une variable interne m_i dont on notera par s_i et r_i la mise à un et la mise à zéro.

Considérons maintenant la gestion de cet ensemble dans le cas d'un grafcet de type II. Celle-ci correspond à l'organigramme de la figure III.42, qui commence par la mise à la valeur initiale des m_i et le remplissage de la liste des transitions validées.

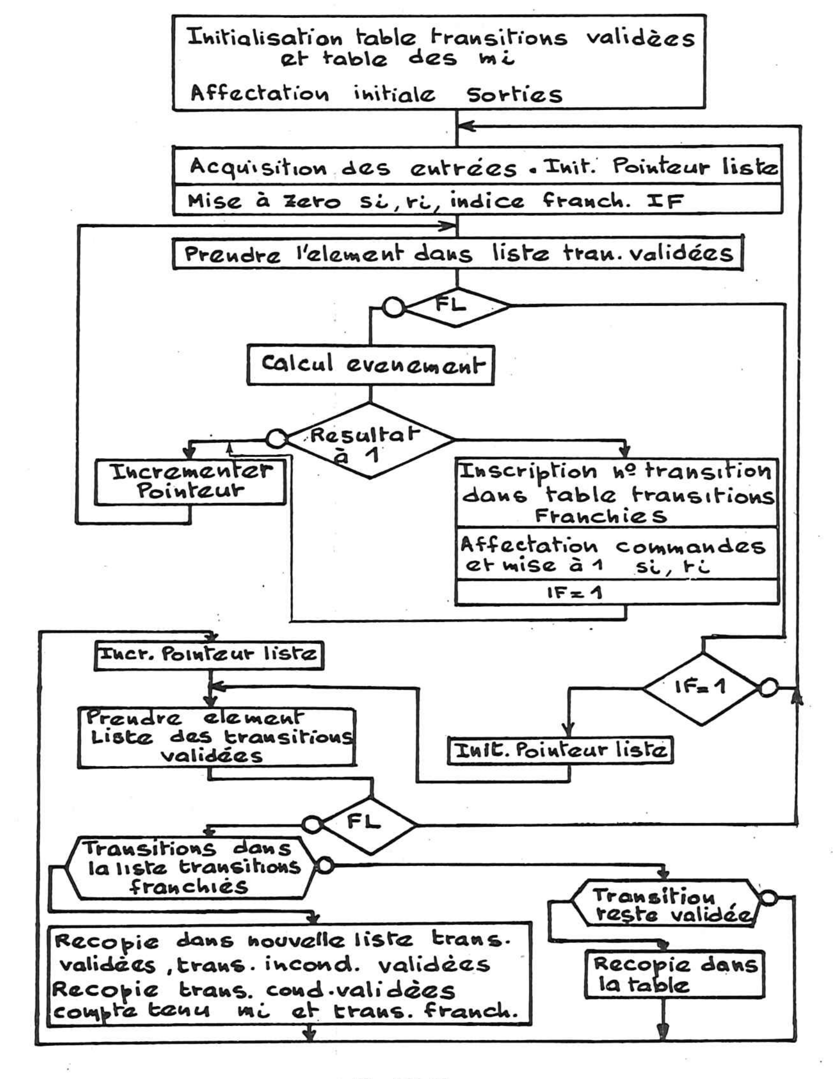


Fig. III.42

Pour chaque transition figurant dans la liste, on examine l'événement associé et, si celui-ci est vérifié, on note le numéro de la transition dans la liste des transitions franchies ; de plus on exécute le combinatoire local lié à la transition et les éventuelles mises à un, ou à zéro, du s_i ou du r_i correspondant. Les s_i et r_i sont mis à zéro en début de cycle. La nouvelle valeur de m_i se calcule en fin de cycle afin de donner la priorité à l'activation.

Une fois tous les franchissements établis, on passe à la détermination de la nouvelle liste des transitions validées en suivant la procédure suivante :

Pour chacune des transitions de la table des transitions validées pour le cycle en cours, on teste s'il y a eu franchissement. Si oui, on recopie dans la nouvelle liste les transitions inconditionnellement validées (autres que jonction). Pour les transitions de type « jonction », on regarde auparavant soit l'état des m_i soit la liste des transitions franchies intervenant dans la condition. Ce recopiage se fait dans tous les cas avec un test destiné à éviter les réécritures. Si la transition n'a pas été franchie, il faut examiner si elle conserve sa validation, en examinant si les transitions concurrentes n'ont pas été franchies. S'il n'en est pas ainsi, on effectue le recopiage.

Enfin, dans le cas où il n'y a aucun franchissement, la procédure est reprise à partir de la même table, après acquisition des entrées.

III.6.3.3. Remarques :

Les méthodes précédentes ont pour but de minimiser le temps de traitement séquentiel, et la quantité d'information à mémoriser.

Dans certains cas de réalisations semi-câblées, le problème ne se pose plus en ces termes, et il devient possible d'associer une mémoire à chaque place et à chaque transition. En fonctionnement asynchrone, le passage à un d'une condition d'évolution entraîne la mise à un de la mémoire liée à la transition correspondante. Deux procédures sont possibles :

Dans la première la mise à un entraîne : la remise à zéro des mémoires correspondant aux places d'entrée et la mise à un des mémoires des places de sortie ; le démarquage des places d'entrée entraîne la remise à zéro des mémoires de transition.

Dans la seconde procédure, la mise à un d'une mémoire de transitions entraîne la mise à zéro des mémoires des places d'entrée, puis, avec un retard ajusté pour éviter les aléas de fonctionnement, la mise à zéro des mémoires de transition suivie de la mise à un des places de sortie.

De telles procédures peuvent facilement s'adapter en mode synchrone.

III.6.4. Réflexions complémentaires

III.6.4.1. Structure orientée programme ou données

Dans l'exposé des méthodes précédentes, nous avons souvent évoqué les notions de structure orientée programme ou données.

Toutefois cette notion est essentiellement relative au niveau de spécialisation du jeu d'instructions de la machine.

La réalisation d'une fonction combinatoire écrite sous forme d'une suite de signes correspondant à l'écriture naturelle d'une équation booléenne peut être, nous l'avons vu dans le premier chapitre, soit considérée à partir d'une table de données qu'un programme va gérer, soit mise en œuvre directement sous la forme d'un programme interprété par un processeur du type analyseur booléen.

Une structure est « orientée données » s'il convient de mettre en œuvre un programme constant destiné à gérer une liste d'informations spécifiques. Mais il est évident que ce programme peut à la limite être interne au processeur.

III.6.4.2. Fonctionnement autosynchrone

Le mot autosychrone souvent employé pour qualifier des réalisations n'indique pas à quelle référence de synchronisme on se rapporte. Il signifie que le traitement s'effectue en synchronisme avec un changement d'entrée ou avec l'apparition d'un événement calculé par ailleurs. Le traitement lui-même peut être synchrone ou asynchrone au sens où nous l'avons défini.

Cette notion d'autosynchronisme, exposée ci-dessus dans quelques cas, peut en fait être adaptée à la grande majorité des méthodes possibles.

III.6.4.3. Traitement par multiprocesseur

Compte tenu des problèmes de parallélisme, de temps d'exécution, ou de capacité technologique d'implantation, il peut être intéressant (sinon obligatoire) d'envisager des traitements par processeurs multiples.

Les possibilités de coopération entre processeurs exploitées jusqu'ici se situent soit au niveau des entrées (pour la détection de variation d'entrée et le calcul d'événements), soit au niveau du traitement en éclatant le réseau en sous réseaux, ou en faisant une décomposition en graphes d'état, soit enfin au niveau de la recherche des places marquées, c'est-à-dire de la validation des transition.

Prenons par exemple un système de détection d'événements coopérant avec un processeur de traitement destiné à traiter un travail de fond en fonctionnement normal. A l'apparition d'un événement, une interruption permet de traiter le programme d'évolution du réseau. Le rôle du système d'entrée est de calculer les événements correspondant à certaines réceptivités (c'est-à-dire à certaines transitions validées) et de générer les interruptions correspondantes. La sélection des interruptions se fait par un masque, calculé à la fin de chaque évolution par le système de traitement en fonction de la validation des transitions.

III.6.4.4. Notion de temps de réponse

Le temps de réponse d'un circuit séquentiel est le temps d'établissement du nouvel état, et de préparation des nouvelles commandes, lors d'un changement d'une variable d'entrée.

Il serait intéressant de pouvoir comparer les diverses méthodes utilisables, mais cela dépend trop de la nature du réseau et de l'importance relative donnée à l'aspect combinatoire et l'aspect séquentiel.

Nous nous contenterons de formuler quelques réflexions.

Dans les cas où on ne dispose pas d'instructions de saut ou si on est limité à l'instruction de saut vers l'avant, il n'est pas possible de jouer sur le fait que la structure de commande n'est réceptive qu'à quelques événements à un instant donné. On doit donc envisager toutes les possibilités. La durée du cycle de traitement est la mesure classique du temps de réponse. Comme il dépend de la longueur du programme, on le ramène à l'exécution d'un millier d'instructions. On aura par exemple 10 ms pour 1K mémoire.

Lorsqu'on dispose d'une instruction de saut général, il devient pratiquement impossible de faire des comparaisons. Entrent en effet comme éléments déterminants dans le temps de réponse, l'aspect interprété ou compilé, la nature de l'interpréteur (programmé, micro-programmé ou même câblé), l'importance du combinatoire local ou général et le degré de parallélisme, c'est-à-dire le nombre de places marquées simultanément. Il est toutefois important de noter que le fait de jouer sur la notion de réceptivité permet d'avoir des performances au moins comparables à celles des déroulements cycliques.

III.6.5. Le logiciel vu de l'utilisateur

Tous les paragraphes précédents avaient pour but de montrer et d'expliquer ce qui se passe dans la machine lors du traitement d'un grafcet ou d'un réseau de Petri, et dans certains cas les programmes ou/et les structures de données ont dû être définis et introduits par l'utilisateur sous la forme présentée. C'est en particulier le cas lorsqu'on utilise directement des machines universelles (avec un jeu d'instructions générales) si le constructeur (ou l'expérimentateur pour les prototypes de laboratoire) n'a pas prévu de programmation des réseaux.

Actuellement il existe peu sinon aucune machine à usage industriel qui dispose d'un langage de description des réseaux de Petri ou des grafcets. Par contre de nombreuses réalisations permettent une programmation plus ou moins sophistiquée des opérations booléennes, des organigrammes et des traitements numériques.

Les microcalculateurs travaillent normalement par assemblage d'opérations utilisant le processeur arithmétique et logique. Aussi pour obtenir des performances logicielles au moins comparables à celles des contrôleurs programmables, il est impératif de créer des langages allant de la stricte transposition d'instructions d'automates programmables jusqu'aux écritures sophistiquées des moniteurs de temps réel.

Les jeux d'instructions d'automates ont eux aussi des puissances très diverses.

La complexité des problèmes conduit à suivre les voies informatiques classiques qui autorisent des descriptions symboliques des variables et des commandes de gestion des tâches.

La description des réseaux de Petri ou des grafcets peut comporter un grand nombre de raffinements classiques en ce qui concerne leurs parties combinatoires. La transcription de la structure du graphe est dépendante de la méthode employée pour l'implantation en machine. Si une décomposition est exigée, chaque sous-graphe (ou branche) est décrit en tenant compte des synchronisations entre les branches.

Dans le cas général, une double description est toujours nécessaire ; l'une portant sur les transitions demande à l'utilisateur la définition des places d'entrée, des places de sortie et de l'événement associé. Dans le cas d'un réseau de type *t*, on y ajoute les commandes. L'autre porte sur les places pour la définition des commandes d'un réseau de type *s*. Mais il est intéressant de profiter de cette deuxième procédure pour faire une vérification croisée demandant pour chaque place, les transitions d'entrée et celles de sortie.

Compte-tenu de la difficulté théorique d'effectuer les vérifications sur des grafcets ou des réseaux de Petri interprétés, les seules aides actuellement possibles sont : soit une analyse syntaxique et une vérification croisée, soit une simulation de l'évolution de l'activité du réseau pilotée par l'utilisateur.

Disposant d'un programme en «langage source», il importe de fabriquer pour la machine un «langage objet» utilisable. Cela peut se faire soit par compilation, soit par interprétation.

La compilation consiste en une traduction dans le langage de la machine, l'exécution se faisant uniquement à partir du code objet, après traduction complète. L'avantage de la compilation est de pouvoir travailler avec n'importe quel type de machine et surtout avec des structures orientées «programme» dont le traitement est généralement plus rapide. Son inconvénient majeur est de rendre pratiquement impossible une relecture du programme traduit, entraînant une difficulté de dépannage et de mise au point.

L'interprétation part du programme source qui est généralement restructuré et allégé des éléments syntaxiques inutiles pour l'exécution. Les modifications sont minimes et n'altèrent pas la description qui peut donc être reprise pour des modifications ou des mises au point. A partir de là, deux techniques sont possibles. Si on désire une structure orientée pro-

gramme, on procèdera par macroinstructions, ce qui implique une certaine forme de description. Par exemple le réseau peut être décrit par des instructions de la forme :

n° d'instruction — SI liste des places d'entrée ET événement,
ALORS liste des places de sorties, FAIRE commandes type t.

Une structure orientée donnée permet une plus grande souplesse de description et en particulier une introduction plus interactive. Les éléments introduits sont restructurés sous forme d'une table comme dans les dernières méthodes présentées. L'accès à cette table permet une relecture aisée de la description. L'interprétation des données introduites est effectuée par le programme de gestion de la table. Les programmes interprétés sont généralement plus lents que ceux qui sont compilés.

IV. LES APPLICATIONS INDUSTRIELLES

Ce chapitre concerne l'utilisation de systèmes programmés de toute nature en vue de la matérialisation industrielle de structures séquentielles de commande. Cette utilisation est basée sur une application directe des méthodes d'implantation des outils de description exposées dans le chapitre précédent, compte tenu des caractéristiques fondamentales du matériel choisi par le concepteur. Nous examinerons successivement l'utilisation des dispositifs spécifiques de la logique cablée programmée, celles des automates programmables et enfin les applications des machines universelles et des microprocesseurs.

IV.1. Les dispositifs spécifiques de la logique cablée programmable

La construction, à l'aide de réseaux logiques ou de circuits de mémorisation, d'un processeur permettant la matérialisation d'une structure séquentielle de commande peut s'effectuer de différentes manières suivant le mode d'adressage retenu et le jeu d'instructions de la machine élémentaire élaborée.

Cette application constitue une extension naturelle des réalisations spécifiques de traitement des fonctions combinatoires et découle généralement sur le plan de la synthèse des méthodes d'implantation conduisant à l'obtention de structures orientées programmes. Chaque réalisation implique la particularisation d'un réseau logique programmable (PLA, FPLA), ou d'un circuit de mémorisation (ROM, PROM, RAM...) dont les caractéristiques (longueur des mots mémorisés, nombre de mots à mémoriser) dépendent principalement du type d'adressage. Sans entrer dans une classification exhaustive des différents modes d'adressage (direct absolu ou rela-

tif, indirect non indexé ou indexé) permettant le passage progressif d'une structure orientée programme à une structure orientée donnée, nous nous limiterons à la description de trois modes d'adressage direct, puis nous illustrerons sur un exemple simple les étapes de construction de dispositifs spécifiques particuliers.

IV.1.1. Les structures à deux adresses et deux types d'instructions

Elles permettent une simulation directe d'un arbre de décision logique par l'utilisation d'une instruction de sortie et d'une instruction de saut. L'instruction de sortie représentée par la figure IV.1a correspond à un rectangle d'affectation et comporte sous une forme codée la variable de sortie à affecter, la valeur de cette variable, et l'adresse de l'instruction suivante. L'instruction de branchement conditionnel représenté par la figure IV.1b correspond à un losange de test et comporte sous une forme codée la variable à tester et les adresses de l'instruction suivante qui dépendent de la valeur de la variable à tester et donc du résultat du test.

type	valeur	sortie	Adresse suivante	a)

type	Variable à tester	Adresse si vrai	Adresse si faux	b)

Fig. IV.1

IV.1.2. Les structures à une adresse et trois types d'instructions

Elles permettent une diminution de la longueur du mot à mémoriser et comportent une instruction de sortie, une instruction de saut conditionnel, et une instruction de saut inconditionnel.

L'instruction de sortie représentée par la figure IV.2.a comporte sous forme codée la variable de sortie à affecter et la valeur de cette variable, l'adresse de l'instruction suivante étant obtenue en incrémentant de 1 le compteur d'adresse.

L'instruction de branchement inconditionnel, représentée par la figure IV.2.b, comporte sous forme codée l'adresse de l'instruction suivante. Elle permet donc un saut en avant ou en arrière.

L'instruction de branchement conditionnel, représentée par la figure IV.2.c, comporte sous forme codée la variable à tester, une valeur de référence 0 ou 1 utilisée pour le test et l'adresse d'une instruction suivante. Si la valeur de la variable est identique à la valeur de référence, l'adresse de l'instruction suivante est généralement celle qui est mémorisée, sinon elle est obtenue en incrémentant de 1 le compteur d'adresse.

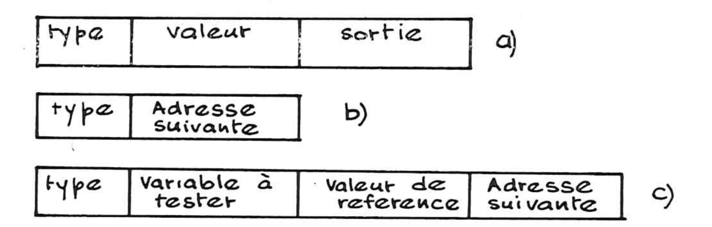


Fig. IV.2

IV.1.3. Les structures à deux adresses et une instruction

Elles permettent une diminution du nombre de mots à mémoriser au détriment de la longueur des mots. L'instruction unique représentée par la figure IV.3.a ou par la figure IV.3.b permet d'englober les notions de sortie et de branchement conditionnel.

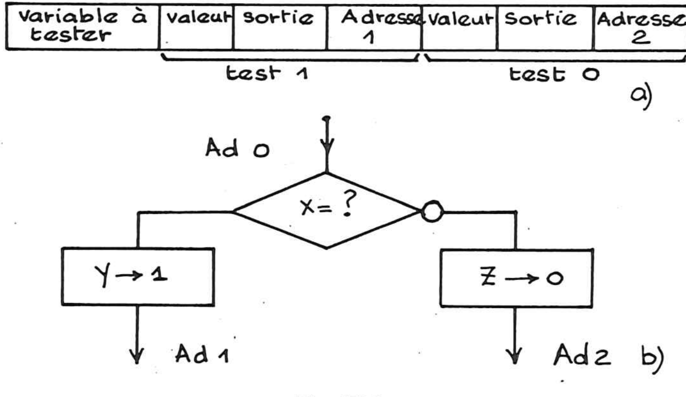


Fig. IV.3

Elle comporte sous forme codée la variable à tester et suivant le résultat du test, l'adresse de l'instruction suivante, la sortie à affecter et la valeur de cette affectation.

Les trois types de machines décrites ci-dessus par leur jeu d'instructions constituent des exemples de machines à mode d'adressage direct. Il est possible de les transformer en machines à incrément en utilisant un mode d'adressage relatif. Dans ce cas les adresses mémorisées dans les instructions décrites plus haut correspondent à des incréments qu'il est nécessaire d'ajouter au compteur d'adresse afin d'obtenir les adresses des instructions suivantes. Cette possibilité de variantes des structures décrites autorise un grand nombre de solutions pour matérialiser les systèmes de gestion plus ou moins complexes d'un réseau logique programmable ou d'un circuit de mémorisation.

La construction d'un système logique câblé programmable répondant au cahier des charges d'un problème d'automatisation peut s'effectuer en utilisant les étapes suivantes :

a) Construction progressive d'un grafcet de type I, ou d'un réseau de Petri de description de la structure de commande résumant le cahier des charges
b) Transformation de ce grafcet en un arbre de décision logique utilisant des rectangles d'affectation et des losanges de test. Le passage à ce graphe d'état se fait par l'obtention préalable du graphe des marquages.
c) Choix d'un jeu d'instructions et représentation tabulaire de l'arbre de décision logique par un ensemble d'instructions
d) Choix d'un code de représentation des informations et écriture de la table à mémoriser
e) Matérialisation de la structure de commande

Nous appliquerons cette méthode au problème de la commande de l'oscillation de la tige d'un vérin.

IV.1.4. Commande de l'oscillation de la tige d'un vérin

On désire en appuyant de manière fugitive sur un bouton poussoir de mise en marche *m* obtenir l'oscillation gdgd.... de la tige d'un vérin pneumatique équipé d'un distributeur électro-pneumatique, fig. IV. 4. La position de la tige du vérin est repérée grâce à deux contacts de fin de course *g* et *d* . La commande en translation du vérin est obtenue par les commandes droite D et gauche G. La tige du vérin est initialement immobile, commandée et positionnée en g. L'arrêt du mouvement est obtenu par action fugitive sur un bouton poussoir à action prioritaire *a*. Tout cycle élémentaire commencé doit être achevé.

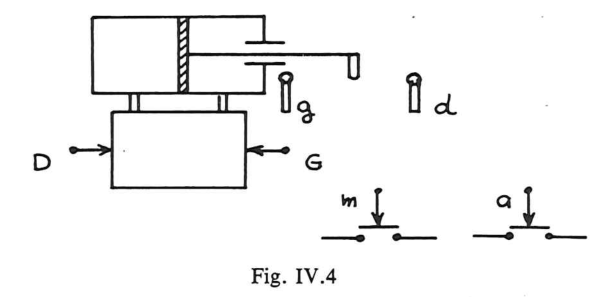


Fig. IV.4

a) Tracé du grafcet

Par application directe de la méthodologie décrite au chapitre II, en partant de l'état initial vérin immobile et non commandé avec sa tige positionnée en *g* il est possible de tracer progressivement le grafcet de type I de la figure IV.5. Dans l'état initial on pourrait également mettre les commandes $\overline{D}$ $\overline{G}$ à cause de la mémorisation par les distributeurs.

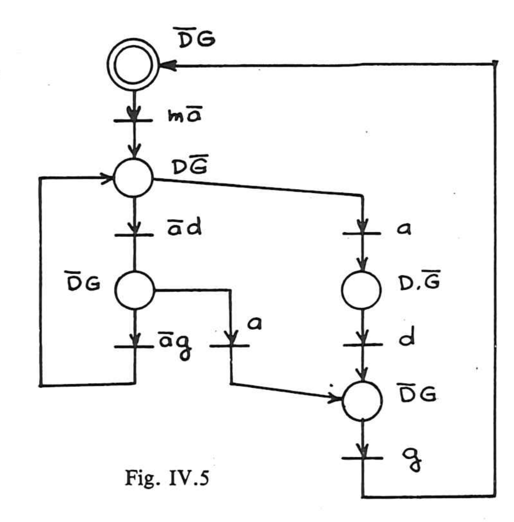


Fig. IV.5

b) Transformation du grafcet de description en un arbre de décision logique. Elle est effectuée directement, car il s'agit d'un graphe d'état, en associant à chaque étape un rectangle d'affectation et à chaque transition un, ou plusieurs losanges de test des variables d'entrée.
Dans le cas où la condition logique associée à une transition correspond à un produit logique, l'ordre des tests élémentaires effectués en série implique une notion de priorité, un seul test pouvant être utilisé en utilisant un opérateur câblé de matérialisation du produit logique.
L'arbre de décision logique représenté par la figure IV.6 correspond au grafcet de la figure IV.5, le test de la variable a prioritaire étant toujours effectué avant celui des autres variables. Dans le cas où les actions sont des incrémentations de compteur, ou des lancements de temporisation, il n'est pas possible de faire le bouclage sur le rectangle d'action. Il convient alors soit de boucler sur le test lui-même, soit sur un rectangle supplémentaire vide d'action.

c) Choix d'une structure à deux adresses et deux instructions : En utilisant le jeu de deux instructions défini en IV.1.1., il est possible de particulariser les instructions en les numérotant sur l'arbre de décision logique. Cette numérotation peut être quelconque et permet la représentation de l'arbre de décision logique représenté sur la figure IV.6 par le tableau équivalent IV.7.

Choix d'une structure à une adresse et trois instructions : En utilisant le jeu de trois instructions défini en IV.1.1., il est possible, compte tenu du

mécanisme d'incrémentation du compteur d'adresse, de particulariser d'une part les instructions de branchement conditionnel et de sortie en les numérotant sur l'arbre de décision logique, et d'autre part d'introduire les branchements inconditionnels supplémentaires. Dans ce cas la numérotation n'est plus quelconque mais doit tenir compte suivant le type d'instruction de la règle de détermination de l'adresse, c'est-à-dire du numéro de l'instruction suivante.

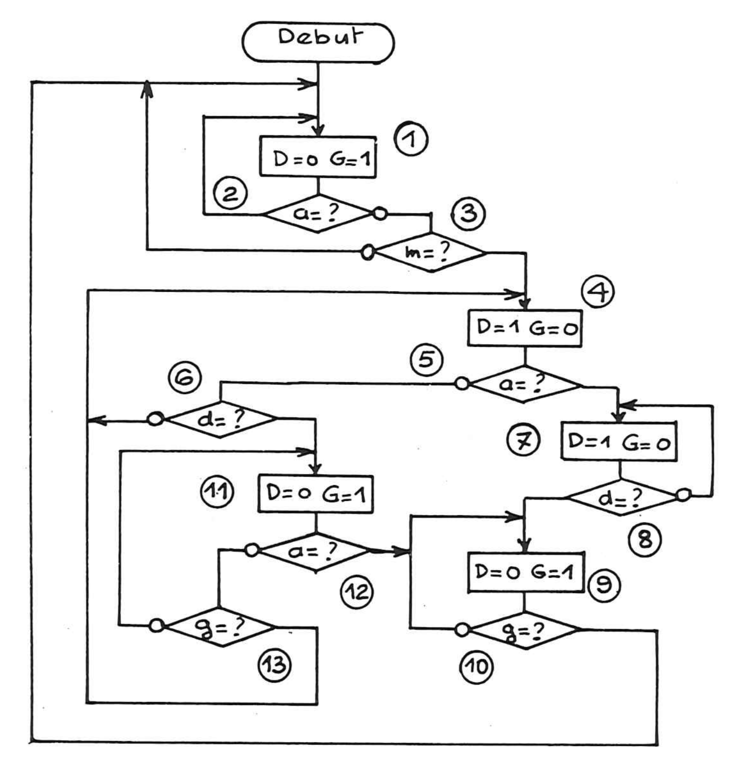


Fig. IV.6

Un arbre de décision logique numéroté et complété est représenté par la figure IV. 8, ou décrit par le tableau équivalent donné par la figure IV. 9.

d) Le choix d'un code de représentation des varaibles d'entrées *a, m, g, d* de la structure de commande nécessite l'utilisation de deux variables x_o, x_1 (fig. IV.10) et celui du numéro de l'instruction, quatre variables i_3, i_2, i_1, i_o (fig. IV.11). Le type d'instruction est codé dans le premier cas avec une seule variable (fig. IV.12) et dans le deuxième cas impose l'utilisation de deux variables *y, z* définissant quatre combinaisons (fig. IV.13).

N° Inst.	Type INSTR.	VAR. α Test	Instr. suiv. α=1	Instr. suiv. α=0	Inst. suiv.	valeur D	valeur G
1	Sort.	–	–	–	2	0	1
2	Test	a	1	3	–	–	–
3	Test	m	4	1	–	–	–
4	Sort.	–	–	–	5	1	0
5	Test	a	7	6	–	–	–
6	Test	d	11	4	–	–	–
7	Sort.	–	–	–	8	1	0
8	Test	d	9	7	–	–	–
9	Sort.	–	–	–	10	0	1
10	Test	g	1	9	–	–	–
11	Sort.	–	–	–	12	0	1
12	Test	a	9	13	–	–	–
13	Test	g	4	11	–	–	–
			Test		Sortie		

Fig. IV.7

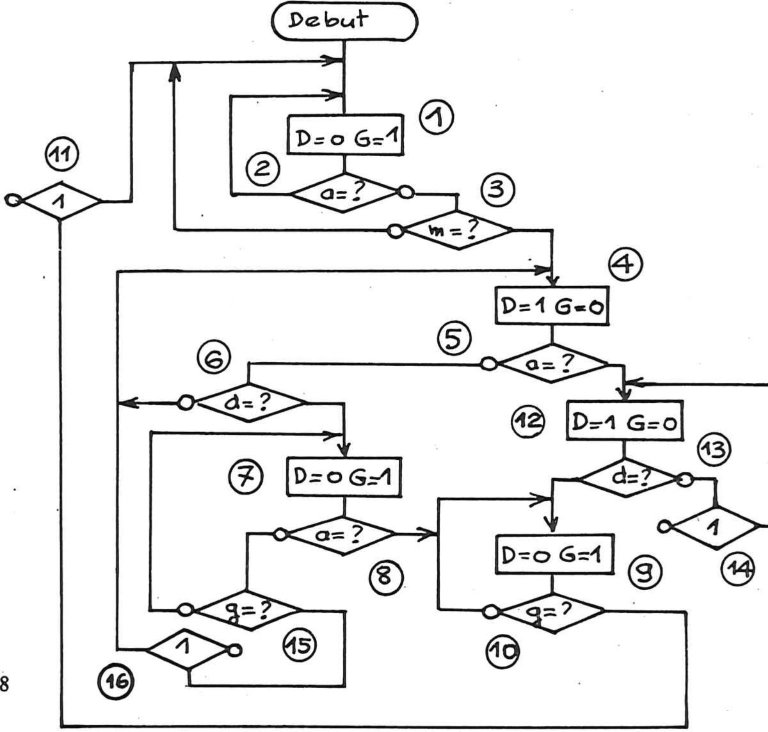


Fig. IV.8

nº INST.	TYPE	VAR. TEST	VAL. REF.	ADR. SUIV.	Sortie valeur D	Sortie valeur G	ADR. SUIV.
1	Sort.	—	—	—	0	1	—
2	Test	a	1	1	—		—
3	Test	m	0	1	—		—
4	Sort.	—	—	—	1	0	—
5	Test	a	1	12	—		—
6	Test	d	0	4	—		—
7	Sort.	—	—	—	0	1	—
8	Test	a	0	15	—		—
9	Sort.	—	—	—	1	0	—
10	Test	g	0	9	—		—
11	Branch. incon.	—	—	—	—		1
12	Sort.	—	—	—	0	1	—
13	Test	d	1	9	—		—
14	Branch. incon.	—	—	—	—		12
15	Test	g	0	7	—		—
16	Branch. incon	—	—	—	—		4

Test (VAR. TEST, VAL. REF., ADR. SUIV.) — Branch. incond. (ADR. SUIV.)

Fig. IV.9

	x_1	x_0
a	0	0
m	0	1
g	1	1
d	1	0

Fig. IV.10

[Nº inst. − 1] ⟷ [i_3 i_2 i_1 i_0]
10 2

Fig. IV.11

	y
Test	1
Sort.	0

Fig. IV.12

	z	y
Test	0	0
Sortie	0	1
Br. incond.	1	0
—	1	1

Fig. IV.13

Les tables à mémoriser sont alors données, compte-tenu des codages choisis par la figure IV.14 pour la structure à deux adresses et deux instructions, et par la figure IV.15 pour la structure à une adresse et trois instructions.

nº INSTRU.					y inst.	Test: Var. α test.		Test: Int. suivante α = 1				Test: Int. suivante α = 0			
nº	i_3	i_2	i_1	i_0	type	x_1	x_0	i_3	i_2	i_1	i_0	i_3	i_2	i_1	i_0
1	0	0	0	0	0	—	—	0	0	0	1	—	—	0	1
2	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
3	0	0	1	0	1	0	1	0	0	1	1	0	0	0	0
4	0	0	1	1	0	—	—	0	1	0	0	—	—	1	0
5	0	1	0	0	1	0	0	0	1	1	0	0	1	0	1
6	0	1	0	1	1	1	0	1	0	1	0	0	0	1	1
7	0	1	1	0	0	—	—	0	1	1	1	—	—	1	0
8	0	1	1	1	1	1	0	1	0	0	0	0	1	1	0
9	1	0	0	0	0	—	—	1	0	0	1	—	—	0	1
10	1	0	0	1	1	1	1	0	0	0	0	1	0	0	0
11	1	0	1	0	0	—	—	1	0	1	1	—	—	0	1
12	1	0	1	1	1	0	0	1	0	0	0	1	1	0	0
13	1	1	0	0	1	1	1	0	0	1	1	1	0	1	0

Sortie: INST. SUIV. — D G

Fig. IV.14

nº INST.	i_3	i_2	i_1	i_0	Type z	Type y	Test: Var. α Test. x_1	Test: Var. α Test. x_0	Test: Ref.	Test: Ad. inst. suiv i_3	i_2	i_1	i_0
1	0	0	0	0	0	1	—	—	—	—	—	0	1
2	0	0	0	1	0	0	0	0	1	0	0	0	0
3	0	0	1	0	0	0	0	1	0	0	0	0	0
4	0	0	1	1	0	1	—	—	—	—	—	1	0
5	0	1	0	0	0	0	0	0	1	1	0	1	1
6	0	1	0	1	0	0	1	0	0	0	0	1	1
7	0	1	1	0	0	1	—	—	—	—	—	0	1
8	0	1	1	1	0	0	0	0	0	1	1	1	0
9	1	0	0	0	0	1	—	—	—	—	—	0	1
10	1	0	0	1	0	0	1	1	0	1	0	0	0
11	1	0	1	0	1	0	—	—	—	0	0	0	0
12	1	0	1	1	0	1	—	—	—	—	—	1	0
13	1	1	0	0	0	0	1	0	1	1	0	0	0
14	1	1	0	1	1	0	—	—	—	1	0	1	1
15	1	1	1	0	0	0	1	1	0	0	1	1	0
16	1	1	1	1	1	0	—	—	—	0	0	1	1

Fig. IV.15

Sortie
Ad. inst. suiv

e) La matérialisation de la structure de commande a été effectuée dans les deux cas (fig. IV.16 et IV.17) à l'aide de mémoires ROM, de circuits multiplexeurs élémentaires, le compteur d'adresse étant réalisé à l'aide de bascules D dont l'initialisation correspond à l'étape initiale du grafcet.

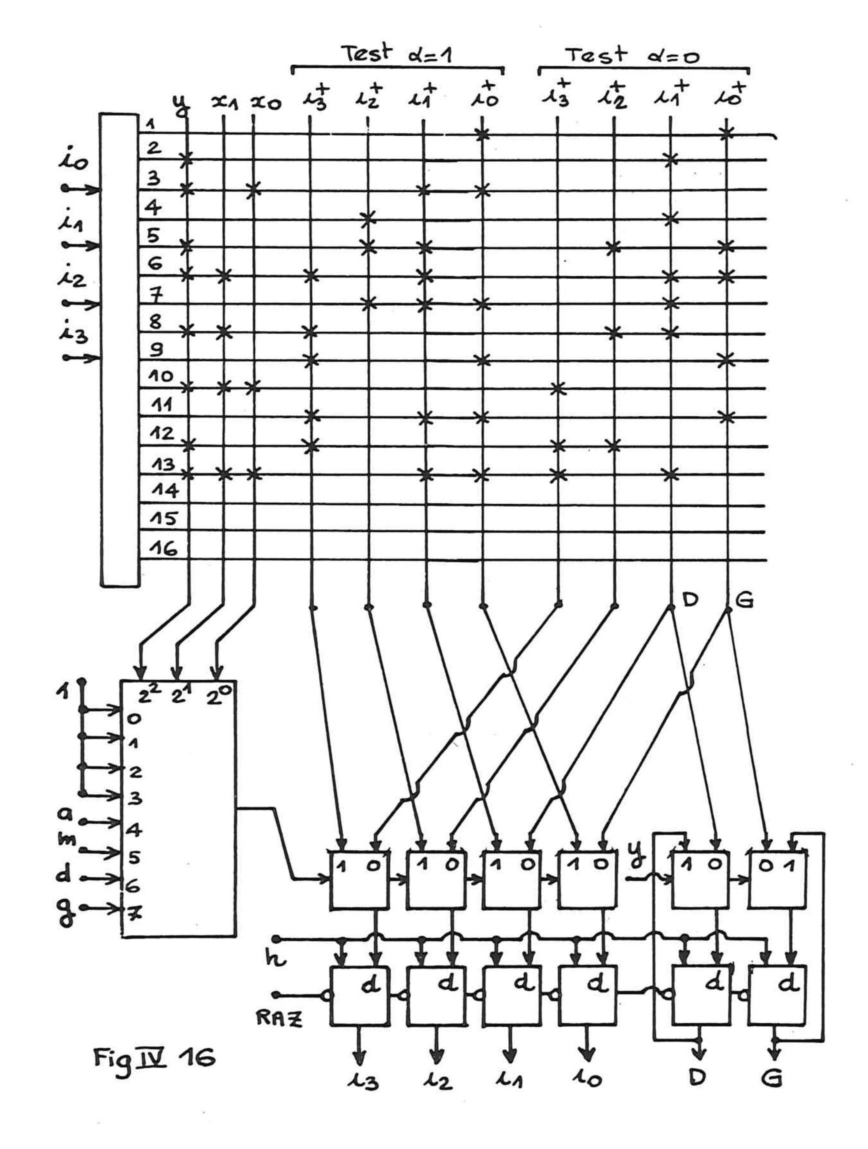


Fig. IV.16

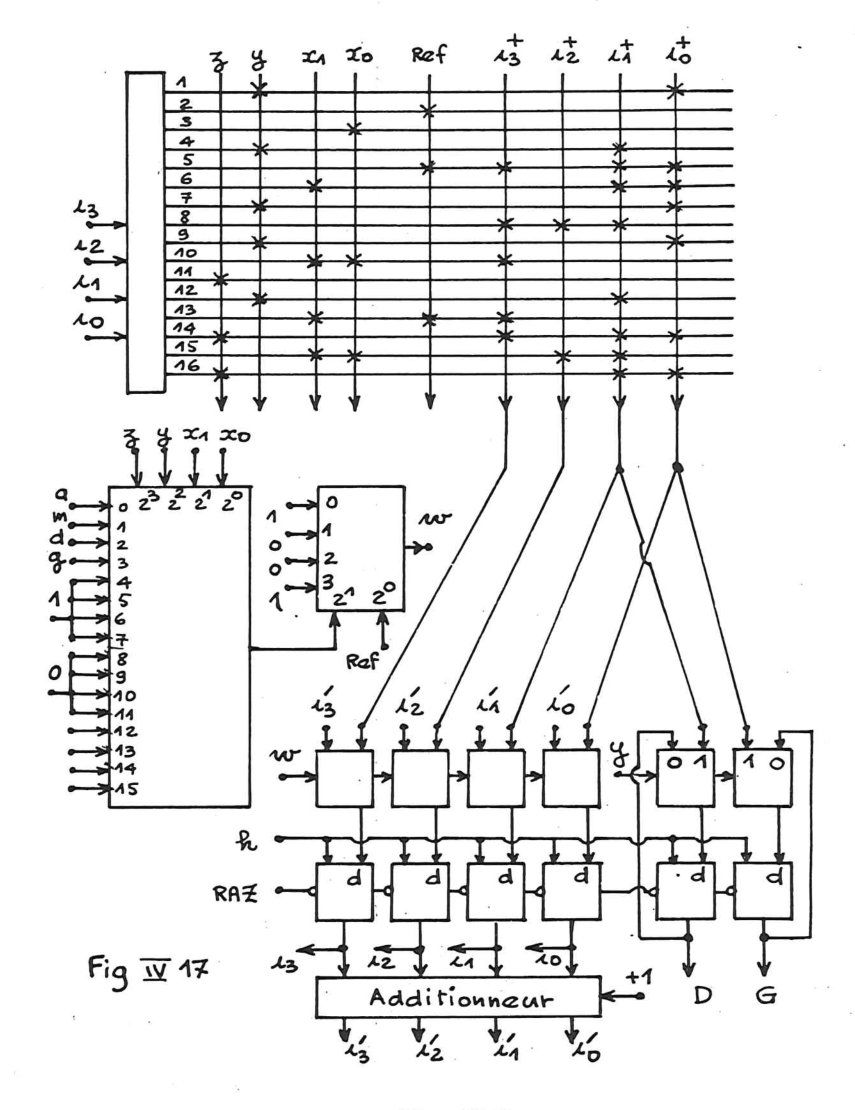


Fig. IV.17

IV.2. Les automates programmables

Les automates ou contrôleurs programmables ont fait l'objet d'une première description dans le chapitre I relatif au traitement des fonctions combinatoires et nous nous limiterons dans ce chapitre à une description

technique de ces machines en insistant plus particulièrement sur l'implantation d'une structure de commande.

Nous examinerons successivement les caractéristiques liées aux langages de programmation, aux dialogues avec le processus de commande ou l'opérateur, et nous terminerons par l'aspect technologique de ces structures.

IV.2.1. Les langages de programmation

Les langages de programmation sont caractérisés par les primitives de base, le type et la durée des traitements, l'existence de fonctions de comptage, de temporisation, et enfin celle d'extensions numériques et de primitives spéciales.

La grande majorité des automates programmables utilise le principe de déroulement cyclique de la mémoire. Quelques-uns dans le haut de la gamme suivent les procédures des calculateurs universels. Dans tous les cas, il importe de se rappeler les notions de traitement, synchrone ou asynchrone, décrites dans le préambule du chapitre III, et de connaître comment s'effectue l'acquisition des valeurs d'entrée et l'affectation des résultats de calcul en sortie, ce qui implique une analyse des instructions spécifiques du traitement séquentiel. En déroulement cyclique en particulier, les entrées sont prises en compte soit globalement en début de cycle, soit dans le calcul, à l'appel de la variable. Ce second cas permet la répétition de certaines séquences liées à des tâches rapides, mais entraîne en pratique la mémorisation en variable interne, ce qui implique un programme plus long et une mémoire plus importante. Les sorties peuvent être appliquées en bloc en fin de cycle, ou au résultat du calcul ce qui peut provoquer des aléas de continuité, ou des contradictions.

Le temps de cycle pour les machines à déroulement cyclique est l'unité de mesure du temps de traitement. Il doit évidemment être ramené à l'unité mémoire, et s'exprime par exemple en ms/K de mots mémoire.

Pour certaines machines permettant de sauter des instructions, il s'agit d'un élément de comparaison. Le problème se pose surtout avec les automates n'utilisant pas le principe de déroulement cyclique. Les constructeurs donnent simplement les temps moyens par instruction mais il est pratiquement impossible d'en tirer des éléments de comparaison tels que le temps de réponse (réaction d'une sortie à une variation d'entrée) ou le temps séparant deux traitements d'un même sous-ensemble.

Les primitives de base, (c'est-à-dire celles destinées au calcul booléen classique), peuvent se classer en six catégories selon les habitudes des utilisateurs. L'écriture sous forme des schémas à relais (ladder diagram) est toujours très en vogue, en particulier aux U.S.A. On trouve également la transcription de logigramme. L'écriture la plus courante suit celle des équations booléennes, avec limitation à un seul niveau de parenthèses.

Certains jeux d'instructions sont tournés un peu plus vers l'informatique, soit par adaptation de l'organigramme, soit par la structuration du processeur autour d'un accumulateur et d'une unité logique. Le dernier type d'écriture, possible sur certaines machines comprenant une pile de traitement, bien que très puissant, n'est pratiquement pas utilisé. Il s'agit de la notation polonaise inverse. Les méthodes d'implantation orientées données sont généralement peu appliquées aux automates.

Le contrôle de processus logiques industriels ne peut se faire sans comptage ou sans temporisation. Une attention particulière doit donc être apportée à ces éléments généralement très peu ou pas explicités par les constructeurs.

Il importe de distinguer les machines qui travaillent avec traitement sur mot de celles pour lesquelles temporisation et comptage se présentent sous la forme de modules dont les entrées et sorties sont décrites par le jeu d'instructions. Ce dernier cas se rencontre sur les machines en bas ou en milieu de gamme. Il présente souvent une moins grande souplesse d'utilisation. En effet les temps de temporisation ou la préassignation des compteurs sont fixés par le programme, et il devient impossible d'opérer des modifications dynamiques, sauf sur certaines machines où il est prévu un boîtier spécial connectable pour la modification en ligne.

Les instructions liées à un module de temporisation, doivent permettre le lancement, la remise à zéro mais aussi le blocage dénoté « gel de la temporisation ». Les modules peuvent parfois être des éléments physiques externes, à valeur préréglée par potentiomètre. Lorsqu'il s'agit de traitement sur mot par horloge en temps réel, il n'y a en principe aucune inquiétude à avoir, car tous les traitements sont possibles, y compris l'accès à tout moment au contenu du temporisateur.

Le nombre d'éléments, les cadences d'horloge et les limites de durée sont les paramètres à examiner.

Le fonctionnement des modules de comptage est beaucoup plus délicat à analyser. Certains de ces modules ne sont en effet que des compteurs, d'autres peuvent à la fois compter ou décompter. Les entrées sont donc : la remise à zéro, le signal d'entrée et l'ordre de comptage ou de décomptage. La sortie indique que la valeur préassignée est atteinte. Il faut veiller particulièrement à ce que les compteurs fonctionnent en cascade au comptage et au décomptage, ce qui n'est pas toujours le cas et ne figure pas toujours clairement dans la notice. Les paramètres sont le nombre de compteurs et la capacité maximale.

Dans le comptage par traitement numérique, on a accès à tout instant au contenu du compteur dont la capacité est fonction de la longueur du mot. En outre certaines instructions numériques comme l'instruction de comparaison par exemple, sont très utiles pour le traitement de certains problèmes.

L'instruction de relais-maître permet la mise en facteur d'une expression pour toute une série d'équations faisant ainsi gagner de précieuses lignes de programme. Les instructions relatives aux sous-programmes ne sont pas fréquentes. Elles facilitent pourtant beaucoup la programmation, et peuvent être mises en œuvre directement au niveau de la description de la structure à implanter.

Les interruptions de programme, quand elles existent, sont hors de la portée de l'utilisateur. Elles sont associées en général aux anomalies du secteur ou du processeur.

Certaines machines de haut de gamme disposent d'extensions numériques. Celles-ci se limitent parfois à l'adjonction d'instructions de comparaison utiles à la réalisation de prédicats liés aux compteurs. Ces extensions peuvent aller jusqu'au traitement complet sur mots, intégrant l'usage des numérotations binaire, décimale, octale, ou de divers modes d'adressage, et même dans certains cas d'instructions de régulation ou de filtrage....

Parmi les primitives spéciales, l'instruction de saut facilite grandement l'implantation d'un langage de description. Certaines extensions non numériques introduisant des instructions spéciales permettent de traiter directement certaines catégories de grafcets, de réseaux de Petri, ou de diagrammes fonctionnels.

IV.2.2. Le dialogue avec le processus

Les entrées-sorties logiques, analogiques, spéciales et les coupleurs logiques doivent être en nombre suffisant et avoir les caractéristiques requises pour assurer l'adaptabilité à des processus de tailles très diverses. En outre, les capteurs et les actionneurs pouvant être de toute nature, l'utilisateur doit disposer d'une panoplie de coupleurs, isolés ou non, pour toutes sortes de types et de valeurs de courants ou de tensions. Les entrées-sorties analogiques n'existent que sur les machines du haut de gamme capables de traiter des informations numériques.

Les entrées-sorties spéciales sont liées à des configurations particulières d'automates. On y trouve par exemple les modules d'interliaison entre automates pour les structures réparties, les modules de liaison à des calculateurs, particulièrement utiles dans les structures hiérarchisées, les connexions type I.E.E.E. 488, ou autres standards, pour les liaisons aux périphériques classiques, les éléments de transmission série (RS 232 C ou CCI TTV24) et de télétransmission.

IV.2.3. Le dialogue avec l'opérateur

Il est assuré par les périphériques standard d'entrée-sortie, la console de programmation et enfin la console de mise au point et de test d'un programme implanté.

En principe ces automates programmables ne sont pas destinés à remplacer les calculateurs universels, c'est pourquoi les périphériques standard sont pratiquement inexistants, sauf pour le haut de gamme avec traitement sur mots, où l'on peut trouver : disque souple (floppy disc), lecteur de rubans, télétype et console de visualisation. En général les périphériques sont intégrés dans la console dont le degré de sophistication dépend de sa richesse en éléments périphériques.

Les périphériques plus courants sont une console de visualisation (en particulier pour figurer les diagrammes à relais), le programmateur de PROM pour les machines travaillant uniquement avec des mémoires mortes, et la platine à cassettes.

En dehors de ces périphériques, la console de programmation a pour rôle essentiel la préparation des programmes et leur introduction dans la mémoire. Il s'agit parfois d'un simple chargement, mais cela peut aller jusqu'à l'analyse syntaxique, l'édition permettant l'insertion et le tassement, la programmation par zone, et même l'écriture symbolique.

Chaque instruction peut être soit interprétée directement et transformée en code machine, soit compilée et traduite.

Le premier cas est en principe moins sophistiqué mais garde au programme en mémoire son sens initial, ce qui facilite les modifications ultérieures, surtout si l'implantation se fait en RAM.

Ces consoles peuvent donc être de caractéristiques excessivement différentes (certaines sont organisées autour d'un micro calculateur) mais l'utilisateur ne peut configurer lui-même sa console. Il est à noter que deux automates d'un même constructeur n'ont pas forcément des consoles compatibles.

Les dispositifs munis de cassettes permettent la mise en place d'une véritable bibliothèque de sous-programmes parfois fournie par le constructeur.

Une part importante du travail de la console est d'assurer la mise au point et les tests. Cela est parfois délégué à un boîtier de test qui permet en outre le test permanent en ligne de la configuration alors que la console inutile en cours de fonctionnement, ne fait les tests que lorsqu'elle est branchée et qu'elle ne sert pas en programmation.

La mise au point d'un programme doit pouvoir faire intervenir des modifications, des avancements pas à pas, des arrêts sur instruction, et la simulation du programme avec forçage des entrées et sorties. Mais la modification en ligne présente un risque, c'est pourquoi l'outil idéal est celui qui permet une mise au point hors ligne avec simulation puis remplacement du programme lors du branchement de la console.

IV.2.4. Caractéristiques technologiques

Elles ont trait à la mémoire, à la nature du processeur et aux alimentations de l'automate.

La mémoire peut être une RAM et nécessiter quelques précautions d'alimentation. Sa nature facilite aussi bien souvent le processus d'implantation qui n'est pas lié à l'utilisation d'un programmateur comme pour les ROM. Les mémoires à ferrites ne sont plus guère employées, quant aux FPLA, elles ne sont qu'un cas particulier sur des produits destinés à des applications spécifiques et figées.

Si le format du mot mémoire est peu important, il convient de connaître la capacité maximale de la mémoire, mais aussi la capacité et la modularité des extensions, afin de savoir si la machine peut s'adapter à des extensions ultérieures. Certaines applications sont en effet très modestes, ou contrôlées par des structures multi-automates. Par contre les gros processus centralisés entraînent parfois la mise en œuvre de plusieurs processeurs à cause du dépassement de capacité mémoire.

La nature du processeur n'est certes pas le point capital à connaître mais permet de situer la machine dans l'échelle technologique des valeurs. Les indications portent sur le type de circuits (TTL, CMOS) mais aussi sur la structure de la réalisation : câblée, microprogrammée, programmée (microprocesseur). Enfin il existe des machines à processeurs interchangeables, ce qui est particulièrement intéressant.

En ce qui concerne les alimentations, on s'intéresse en fait à ce qui se passe en cas de coupure de celles-ci, même si les mémoires sont de type ROM. Il est en effet important, industriellement, de savoir si la sauvegarde du programme est possible, partielle ou totale, et s'il existe éventuellement une procédure automatique de reprise, soit à l'endroit d'arrêt, soit après une réinitialisation.

IV.2.5. Méthodes d'implantation des grafcets sur automates programmables.

Les caractéristiques des automates évoquées ci-dessus nous montrent qu'il existe quatre grandes classes d'automates :

— Les automates qui ne disposent pas d'instructions de saut, pour lesquels il est seulement possible d'employer les méthodes indiquées au chapitre en première catégorie.

— Les automates qui disposent d'une instruction de saut vers l'avant. Cette instruction, qui fait partie du jeu d'instructions de la majorité des automates du commerce, confère des facilités d'implantation du grafcet, et autorise l'utilisation des méthodes de la deuxième catégorie ;

— Les automates qui disposent d'instructions spéciales orientées grafcet, et pour lesquels il est très important de bien analyser le caractère synchrone ou asynchrone de la méthodologie d'implantation ;

— Les automates, constitués souvent de microprocesseurs, qui acceptent un langage du type organigramme, et pour lesquels il est possible de travailler avec des méthodes plus générales.

Dans tous les cas, les implantations sont orientées « programme ». De la richesse en extensions numériques dépend la facilité d'association aux transitions de prédicats complexes, et l'adaptation à des taches pouvant comprendre des traitements analogiques.

IV.3. Les machines universelles et les microprocesseurs

Par la variété des structures rencontrées tant sur le plan matériel que sur le plan logiciel, il est difficile de décrire les diverses matérialisations des structures séquentielles industrielles sur machines universelles ou microprocesseurs.

Il faut toutefois noter que les microprocesseurs permettent une parfaite adaptation aux besoins, grâce à la gradation dans la complexité des structures depuis le « monochip », limité en mémoire et en lignes d'entrée-sorties, jusqu'au microordinateur.

Nous nous limiterons dans ce paragraphe à de brèves généralités sur le matériel en renvoyant le lecteur pour plus de détails aux ouvrages spécialisés sur les dispositifs de commande en temps réel.

Il n'y a pas de différence de concept entre microprocesseurs et machines universelles. Toutefois sur le plan de la réalisation, dans les premiers le concepteur a la possibilité de choisir la structure la meilleure, tandis que pour les secondes le choix est plus restreint.

Cette constatation résulte du fait que les microprocesseurs peuvent être regroupés en véritables familles de composants.

Il est possible de résumer comme suit les critères du choix d'un microprocesseur :

— Format de données 1-4-8-12-16 bits
— Jeu d'instructions et types d'adressage
— Structure de l'unité centrale
— Vitesse de traitement
— Possibilités d'interruption
— Propriétés des interfaces
— Caractéristiques de la mémoire (si elle est interne)
— Tension d'alimentation
— Disponibilité des composants de la famille

— Matériel et logiciel d'aide à la programmation et au développement.

— Coût des composants

Parmi les éléments de cette liste non exhaustive, il convient de souligner quelques paramètres particulièrement importants.

En dehors de quelques processeurs de format 1 bit, destinés à servir d'unité centrale d'automate programmable, le format doit être bien adapté au traitement des mots de comptage, de temporisation, de codage numérique.

Le format 8 bits, bien adapté au traitement de caractères ou d'information binaire BCD n'est pas suffisant pour couvrir l'étendue nécessaire à certains comptages et à la temporisation. De plus il ne donne pas une précision suffisante pour certains codages. Un disque codeur à 8 pistes ne pourrait fournir une information qu'à ± 0,7°. Il est toutefois possible dans ce cas de travailler sur deux mots en augmentant le temps de traitement.

En logique industrielle, les problèmes nécessitent un traitement sur bits. Il est donc particulièrement utile de disposer d'un jeu d'instruction permettant un accès immédiat à un bit d'un mot si l'on ne veut pas allonger le traitement. En effet le temps est un impératif primordial et il est souvent difficile d'évaluer et de comparer les diverses méthodes proposées au chapitre III, le temps de réponse étant en fait fonction du degré de simultanéité, mais surtout très dépendant de la quantité de calculs combinatoires liés aux transitions et aux actions. On aura toujours intérêt à viser un matériel à grande vitesse de traitement. L'expérience montre que c'est là que réside la limitation de l'usage des microprocesseurs.

La mise en œuvre de structures à multiprocesseurs est certainement une solution à ce problème et s'il est actuellement aisé d'implanter des graphes décomposés, il reste beaucoup de travail à faire sur les méthodes à suivre, et sur les aides à l'implantation.

IV.4. Mise en œuvre des méthodes

Disposant d'une instruction de saut générale, les méthodes de la troisième partie du chapitre III sont les plus intéressantes car elles utilisent pleinement la notion de réceptivité.

Considérons par exemple la méthode du paragraphe 3.6.2 orientée «données», portant sur les places, et essayons de définir la structure la mieux adaptée.

Le programme moniteur gestionnaire de la table spécifique des données, ainsi que cette table spécifique à chaque application sont placées en mémoire morte. Un certain nombre de cases de mémoire vive sont nécessaires

pour créer les images des entrées, des sorties et des variables internes. Comptetenu de la structure de certains modules d'E/S, les images des entrées et des sorties peuvent être situées directement dans les registres internes de ces modules. Par contre pour les variables internes, nécessaires au traitement combinatoire et à la table des pointeurs courants, il est intéressant d'utiliser une mémoire du type RAM. Cette mémoire RAM étant de petite dimension, il y a quelquefois intérêt à utiliser un microprocesseur muni de registres internes en nombre suffisant pour ne pas avoir à placer de module mémoire RAM. Toutefois, s'il est nécessaire de mettre en œuvre des opérations de comptage ou de temporisation, un tel module est bien utile.

Le meilleur moyen pour faire une génération du temps est d'utiliser le système d'interruption piloté par une horloge synchronisée sur le secteur.

On aboutit ainsi à la structure classique représentée par la figure IV.18.

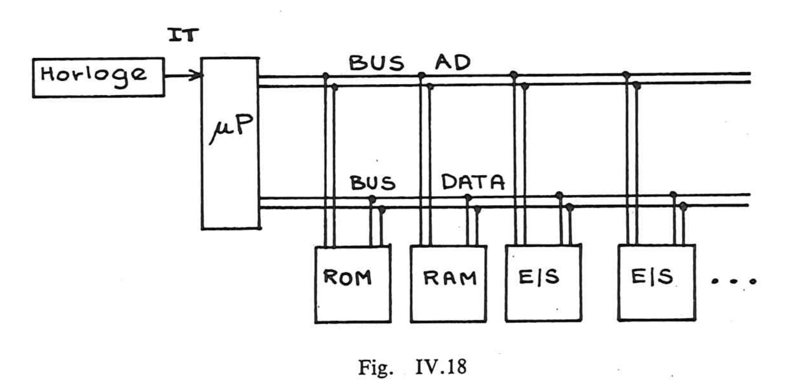


Fig. IV.18

Lorsque le nombre d'entrées/sorties n'est pas élevé, l'ensemble de cette structure peut être incluse dans un monochip, assurant une grande puissance de commande sous un tout petit volume. En général, les mises en œuvre sont orientées «données», toutefois en «monochip», compte tenu de la petite taille de la mémoire, on a intérêt à prendre des implantations orientées «programmes» comme celle du paragraphe 3.6.2.

V. BIBLIOGRAPHIE

Nous nous sommes volontairement limités dans cette courte bibliographie aux références qui ont été pour nous d'un apport certain, et peuvent permettre au lecteur d'approfondir une question, en excluant les éléments trop marginaux ou trop théoriques.

La classification des éléments de bibliographie a été effectuée en distinguant livres ; actes de congrès ; rapports de base ; thèses ou rapports scientifiques et enfin les articles.

V.1. Livres

M. BLANCHARD - « *Comprendre, maîtriser et appliquer le Grafcet* » Cepadues, ed. oct. 1979.

M. BLANCHARD, E. DACLIN, *Synthèse des systèmes logiques,* Cepadues 1976.

MM. BOSSY, BRARD, FAUGERES, MERLAUD, *Le Grafcet, sa pratique et ses applications,* Educalivre ed. oct. 1979.

C.R. CLARE, *Designing logic systems using state machines,* Mc Graw Hill 1973.

P. GIRARD, P. NASLIN, *Construction des machines séquentielles industrielles,* Dunod 1973.

G. MICHEL, C. LAURGEAU, B. ESPIAU, *Les automates programmables industriels,* Dunod 1979.

S. THELLIEZ, *Pratique Séquentielle et réseaux de Petri,* Eyrolles 1978

V.2 Actes de Congrès

Congrès AFCET- *Automatismes logiques,* Paris, décembre 1976.

AFCET réseaux de Petri, Paris, mars 1977 (tendance théorique).

Les méthodes modernes d'étude et de réalisation des automatismes, colloque AFCET-SEE, Gif-sur-Yvette, 2-3 février 1978.

V.3. Documents de base

Rapport AFCET : «*Commission de normalisation du cahier des charges d'un automatisme logique*» groupe de travail «systèmes logiques». Automatique et informatique industrielles n° 61-62, nov., déc. 1977.

Rapport ADEPA : *Le Grafcet diagramme fonctionnel des automatismes séquentiels,* avril 1979.

V.4 Thèses et rapports scientifiques

C. ANDRE - *Sur une méthode de conception assistée par ordinateur des systèmes logiques à évolutions simultanées.* Thèse 3e cycle, Nice, juin 1975.

M. AUGUIN - *Conception des systèmes de commande à l'aide de réseaux logiques programmables.* Thèse 3e cycle, Nice, novembre 1978.

M. COURVOISIER - *Description et réalisation des systèmes de commandes asynchrones à évolutions simultanées* - Thèse de Doctorat es Sciences, Toulouse, février 1974.

J. DEFRENNE - *Implantation de réseaux de Petri sur automate biprocesseur à haute sûreté de fonctionnement* - Thèse 3e cycle, Lille, juin 1979.

G. DELORY - *Automate programmé par réseaux de Petri,* Mémoire CNAM, Lille, mai 1978.

M. HACK - *Petri net languages - technical report 159 MIT,* mars 1976.

S.S. PATIL - *An asynchronous logic array Technical memo 62 - Project MAC - MIT,* mai 1975.

C. RAMCHANDANI - *Analysis of asynchronous concurrent systems by Petri nets MAC TR-120,* février 1974 - MIT

J. RENALIER - *Analyse et simulation en langage APL de systèmes de commande décrits par des réseaux de Petri* - Thèse 3e cycle, Toulouse 1977.

J. SIFAKIS, J. PULOU, M. MOALLA - *Réseaux de Petri synchronisés LAG.* RR n° 80 Université de Grenoble, sept. 1977.

SILVA SUAREZ - *Contributions à la synthèse programmée des automatismes logiques* - Thèse docteur ingénieur, Grenoble, juin 1978.

R. VALETTE - *Sur la description, l'analyse et la validation des systèmes de commande parallèle.* - Thèse doctorat d'état Toulouse, nov. 1976.

M. ZACHARIADES - «*MAS réalisation d'un langage d'aide à la description et à la conception des systèmes logiques*» - Thèse de 3e cycle, Grenoble 1977.

V.5. Articles de Revue

S.B. AKERS - *Binary decision diagrams. IEEE Trans. on computers* vol c-27 n° 6, juin 1978.

G.M. BEDNAR, J.H. TRACEY, ACDL - *An asynchronous circuit design language. IEEE Trans. on computers,* septembre 1974.

M. BLANCHARD - *Automatismes logiques «Grafcet ou réseau de Petri» Le nouvel Automatisme* n° 6, mai 1979, pp. 45-52.

M. COURVOISIER - *An asynchronous logic array for the realisation of logic systems with concurrency. Electronics letters,* 16 février 1978, vol. 14 n° 4.

L. DADDA - *On the simulation of Petri nets as a control tool. Euromicro,* janvier 76, vol. 2 n° 1, pp. 38-45.

L. DADDA - *Petri nets for controlling and synchronizing complex asynchronous operations, Euromicro,* 1977.

G. FANTAUZZI, A. MARSELLA, *A proposal for programmable and modular logical circuits, Automatisme Tome XVIII,* n° 12, déc. 1973.

G.I. IVANOV - *The universal program method and its application for implementing boolean functions using microcomputers - Euromicro journal* 4 (1978) 87-91.

J.V. LANDAU - *State description techniques applied to Industrial Machine control - IEEE. Trans.* on comp. Feb. 1979.

P.E. LAUER, R.H. CAMPBELL - *Formal semantics of a class of high level primitives for coordinating concurrent processes - Acta Informatica 5* - 297-332 (1975)

K.C. LEUNG, C. MICHEL, P. LEBEUX - *Logical systems design using PLAS and Petri nets - programmable hardwired systems, IFIP Information processing* 1977.

D. MANGE - *Arbres de décision pour systèmes logiques câblés ou programmés. Bulletin ASE/UCS* t 69 (1978) n° 22 pp 1238-1243

J.P. PARSY, J.M. TOULOTTE - *A method for decomposing interpreted Petri nets and its utilization. Digital Processes 5 (1979) 3-4.*

J.L. PETERSON - *Petri nets. Computing surveys* - vol. 9 n° 3, sept. 77.

J.L. POKOSKI - *Software analysis for combinatorial logic - Computer design,* juin 1978.

F. PRUNET, J. DUMAS - *Introduction à la modélisation naturelle des structures de commande : l'organiphase - Rairo J2,* juillet 1974, pp 45-75

H.A. SHOLL, S.C. YANG - *Deisng of asynchronous sequential networks using read only memories - IEEE Trans on computers* vol. C 24 n° 2, February 1975.

B. TACONNET, B. CHOLLOT - *Programmation du Grafcet sur automates à langage logique, à relais ou booleen - Le nouvel automatisme* n° 4 janvier, février 1979, pp 41-45.

J.M. TOULOTTE - *Réseaux de Petri et automates programmables - Automatisme* Tome XXII n° 7-8 juillet-août 1978, p 200-211.

J.M. TOULOTTE - *La fiche Technique idéale d'un automate programmable - Le nouvel automatisme,* Sept.-Oct. 1979.

L. TOURRES - *Une méthode nouvelle d'étude des systèmes logiques et son application à la réalisation d'automatismes programmés - RGE T 85,* n° 3, mars 1976.

R. VALETTE - *Sur la description, l'analyse et la validation des systèmes de commande parallèle.* Thèse doctorat d'état. Toulouse, nov. 1976.

Dépôt légal: Août 1982 – N° d'Éditeur 3831
Imprimerie de la Manutention, MAYENNE